INTRODUZIONE AI MICROCOMPUTER

JACKSON

VOLUME 0
IL LIBRO DEL PRINCIPIANTE

di ADAM OSBORNE

INTRODUZIONE AI MICROCOMPUTER

VOLUME 0
IL LIBRO DEI PRINCIPIANTI

La Adam Osborne and Associates Inc. ringrazia per il prezioso lavoro svolto nella stesura dell'edizione originale il seguente staff redazionale:

Graphics:	George Vrana Karen de Robinson
Cartoons:	**Bruce Mishkin**
Typography:	Vicki Mitchell Penny Lawrence
Proofreaders:	Sally Kusch Laurie Battle
Typist:	Marcia McCuen
Coordinator:	Mary Borchers
Printed by:	Rotary Offset Printers San Francisco, California

Le fotografie pubblicate in questo libro sono state cortesemente concesse da:

The Byte Shop
Digital Equipment Corporation
Heath Company
Heuristics, Inc.
Processor Technology Corporation
Qume Corporation
Technical Design Labs
Vector Graphics, Inc.

La Jackson Italiana Editrice ringrazia per il prezioso lavoro svolto nella stesura dell'edizione italiana le signore Francesca di Fiore e Rosi Bozzolo.

Stampato in Italia da:
S.p.A. Alberto Matarelli - Milano - Stabilimento Grafico

SOMMARIO

SOMMARIO (continua)

SOMMARIO (continua)

CAPITOLO **PAGINA**

INDICE DELLE FIGURE

INDICE DELLE TABELLE

INDICE ANALITICO

INDICE ANALITICO (continua)

INDICE ANALITICO (continua)

INDICE ANALITICO (continua)

INTRODUZIONE

Questo corso è stato scritto per quei lettori che non sanno niente (o quasi!) su calcolatori ed elaboratori.

Ci si rivolge quindi a due categorie di persone:

1) a quelli di voi che hanno un concreto interesse ad imparare come usare i calcolatori;
2) a tutti coloro (ormai sono tanti) che, volenti o nolenti, devono vivere (per motivi professionali) con i calcolatori, e che quindi è meglio ne sappiano qualcosa.

Questa prima parte del corso spiegherà superficialmente come lavorano i calcolatori e cosa possono fare.

Ma se la maggior parte di voi può vivere felicemente la sua vita senza mai programmare un calcolatore, o addirittura senza toccarlo, perchè dovrebbe leggere queste pagine e digerirne il contenuto?

La risposta è molto semplice: a livello sia di hobby che di industria, sia di amministrazione che di esecuzione lavori, i microcalcolatori stanno facendo passi da gigante ed entrando nella vita di tutti i giorni (negli USA esistono già i grandi magazzini esclusivamente dedicati agli "hobby's microcomputers").

Vediamo subito l'affermazione del computer nella società.

I calcolatori, come l'automobile e l'elettricità, stanno diventando parte integrante della vita di tutti i giorni, in ogni società industriale, nella quale ogni ora della normale vita di ciascuna persona può essere interessata, e anche più volte, dal contatto con un calcolatore.

Il vostro nome è probabilmente su una qualche lista di vendite per corrispondenza o di opuscoli pubblicitari in genere o, più semplicemente, delle riviste cui siete abbonati; il tutto è gestito da un calcolatore e dietro il... suo consenso voi riceverete questa posta. Infatti un grosso elaboratore può stampare migliaia di etichette/indirizzo in un minuto; se un dattilografo (o altro compositore) dovesse "battere" tali indirizzi, i tempi ed i costi sarebbero enormi.

Le "carte di credito" esistono perchè esistono i calcolatori; se la contabilità relativa dovesse essere gestita a mano, i movimenti di accreditamento renderebbero tale servizio assolutamente ineconomico.

Buona parte dei servizi di prenotazione viaggi (specie all'estero) può avvenire con celerità e sicurezza solo avvalendosi di calcolatori allacciati a linee telefoniche o a ponti radio.

E tuttociò naturalmente è surclassato (sotto l'aspetto dell'importanza) dall'amministrazione di industrie piccole e grandi, senza contare l'esecuzione di lavori con macchine utensili, la gestione dei magazzini e tutte le operazioni connesse, e via via dal funzionamento delle banche, degli enti statali e degli organismi internazionali. E il futuro... non è ancora cominciato!

Il censimento del 1950 negli USA è stato possibile usando l'ENIAC I, da alcuni considerato come il primo computer commerciale del mondo. ENIAC I costò più di mezzo milione di dollari (e dollari del 1950).

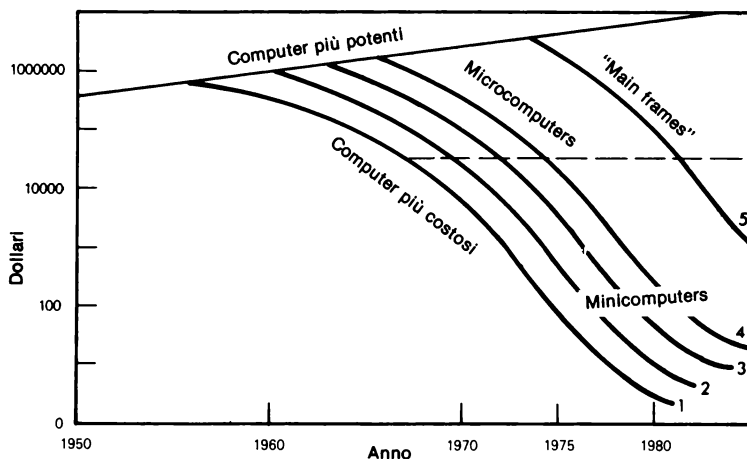
Oggi voi potete acquistare pressoché la stessa "capacità di calcolo" per... 10 dollari (e dollari del 78). Alla fine degli anni 50 e agli inizi degli anni 60, cominciano ad elaborare dati per grandi compagnie (le uniche che se li potevano permettere) calcolatori che costavano un milione di dollari (o più). Un sistema computer equivalente costa oggi dai 2000 ai 3000 dollari, accessibile quindi a qualunque supermercato.

E infatti, i sistemi a calcolatore sono oggi talmente poco costosi da essere usati per i giochi video; giocattoli per bambini, timer per cucine o lavanderie; non è certo troppo distante un futuro nel quale le auto potranno disporre di qualche computer che controlli il motore e la strumentazione di bordo.

Tutti abbiamo potuto constatare come l'elettronica ha agito sulle calcolatrici e sugli orologi, che sono ora pilotati da dispositivi che, in realtà, non sono altro che piccoli computer, o elaboratori.

L'industria della musica potrà presto essere rivoluzionata dall'accoppiata computer-elettronica. E' probabile che nel giro di una decina d'anni dischi e nastri vengano sostituiti da piccole memorie elettroniche.

C'è comunque da dire che negli ultimi 25 anni c'è stata tutta una serie di invenzioni che hanno drasticamente ridotto il costo dei circuiti di conteggio, e quindi del computer. Ma, sorprendentemente, si sono verificate poche varianti nei concetti fondamentali che costituiscono i computer stessi. Quindi, a seguito di ogni nuovo salto tecnologico, noi abbiamo costruito gli stessi calcolatori della volta precedente ma con costi molto più bassi; e abbiamo anche costruito nuovi, più potenti computer allo stesso prezzo dei vecchi.

Il grafico illustra l'andamento cui ora abbiamo accennato.

MINICOMPUTER E MAINFRAME

Con l'enorme differenza fra prezzi e prestazioni di computer che comparvero durante gli anni 60, era inevitabile che si verificasse qualche stratificazione a differenziare i prodotti. Attorno al 65 gli elaboratori meno costosi cominciarono ad essere identificati col termine di "minicomputer"; il prefisso "mini" derivò dal fatto che questi nuovi calcolatori oltre ad essere più economici dei loro predecessori, avevano anche dimensioni fisiche più modeste.

Per differenziarli da questi, i computer più potenti e costosi vennero chiamati "mainframe", che tenteremo di tradurre come complesso elaboratore.

Ora, se chiedere a 10 persone (che se ne intendono, almeno un po') di definire la differenza esistente fra un minicomputer ed un mainframe, quasi certamente riceverete in risposta 10 definizioni totalmente diverse.

In realtà, le uniche differenze fra mainframe e minicomputer sono il prezzo e l'esecuzione.

I minicomputer sono molto meno costosi e, in genere, meno potenti (per quanto i più complessi prodotti venduti come minicomputer sono più potenti dei mainframe più modesti).

I mainframe sono usati come elaboratori di dati commerciali o scientifici, mentre i minicomputer sono venduti in varie esecuzioni e per usi più ampi e svariati.

In effetti, le differenze fra le due versioni sono spesso piuttosto sottili, e l'elemento determinante può essere addirittura solamente la denominazione secondo la quale il prodotto viene venduto.

MICROCOMPUTER

In modo abbastanza analogo, la storia si è ripetuta con l'avvento del microcomputer. Attorno al 72, cominciarono ad apparire prodotti a costi molto più bassi dei soliti computer, e furono denominati "microcomputer".

Il prefisso si riferiva all'aspetto molto più ridotto delle apparecchiature, se paragonato a quello dei minicomputer, esattamente come il prefisso "mini" si era basato sulla differenza di mole rispetto al mainframe.

Naturalmente, ancora una volta esiste una sostanziale interferenza fra i prodotti indicati come microcomputer e quelli indicati come minicomputer, esattamente come nel caso precedente.

C'è comunque da dire che i microcomputer, per il loro limitato ingombro e prezzo, vengono usati per applicazioni che non avrebbero mai potuto adottare un minicomputer.

A questo punto, non serve andare più oltre nello studio delle differenze esistenti fra "mainframe", "minicomputer" e "microcomputer"; oltretutto, non esistono differenze veramente fondamentali di impostazione, se non nelle dimensioni, potenzialità e prezzo.

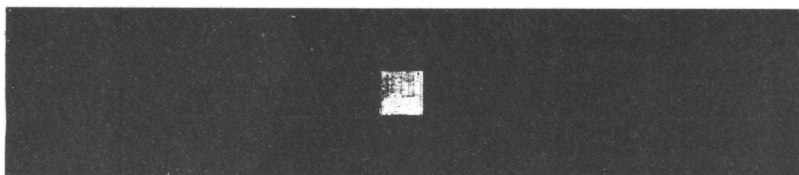
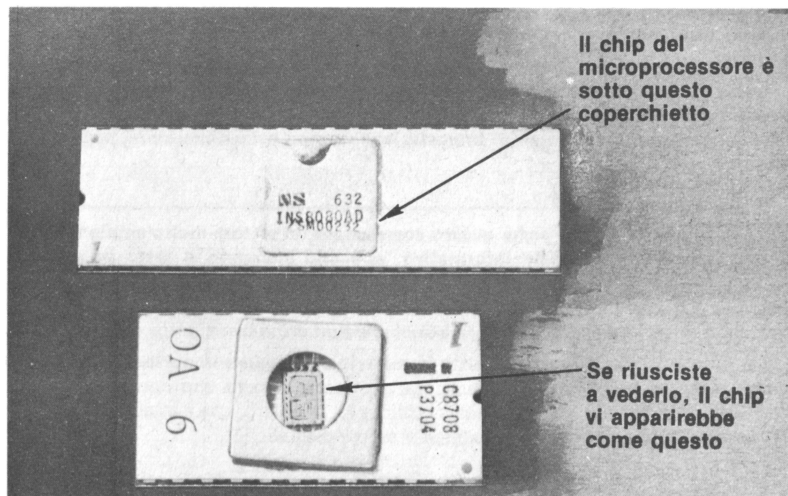
LA STAMPA DEL TESTO

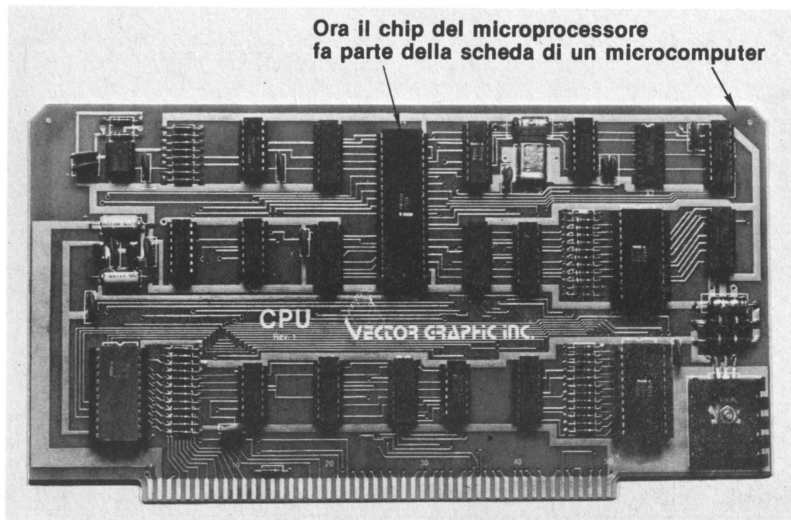
Poichè questo corso copre un settore molto ampio di materiale informativo, abbiamo stampato il testo in parte in grassetto; in parte in chiaro; lo scopo di avere due aspetti di stampa è quello di permettervi di viaggiare spediti sulle parti che avete facilmente acquisito e di soffermarvi sulle informazioni che non avete ben capito.

Infatti il testo in nero incorpora tutta la materia di maggiore importanza. Quando vi trovate ad aver a che fare con qualcosa, stampato in nero che non avete capito, allora sarete costretti a leggere la parte esplicativa, in chiaro, che è quella che fornisce le ulteriori informazioni utili per la miglior comprensione.

Capitolo 1
LE PARTI CHE COSTITUISCONO IL TUTTO

LSI:
INTEGRAZIONE A LARGA SCALA

L'età dell'industria del computer per il mercato del consumo è il risultato di una nuova tecnologia che permette di stivare decine di migliaia di microscopici circuiti elettronici in un rettangolino che in genere ha lati non superiori a 2 ÷ 3 mm. Questa tecnologia è indicata con l'abbreviazione LSI, che significa "integrazione a larga scala"; qui sotto è riprodotto un dispositivo LSI, in dimensioni reali.

Al giorno d'oggi è possibile realizzare, entro un singolo dispositivo LSI di queste dimensioni, tutti i circuiti necessari per creare il "cervello" di un semplice calcolatore. Questo dispositivo LSI è chiamato microprocessor. Il microprocessore (che sarebbe più esatto chiamare microelaboratore) è quindi il calcolatore interno di ogni sistema di microcalcolatore.
Qui sotto riproduciamo un microprocessore, sempre in dimensioni reali.

Ora che sappiamo che aspetto ha un microprocessore, possiamo anche dimenticarcelo. Vi ritorneremo sopra solamente più avanti, in quanto esso non rappresenta che una parte del microcalcolatore, alla cui costituzione più generale ora ci dedichiamo.

Ora il chip del microprocessore
fa parte della scheda di un microcomputer

Il "microcomputer", che finora abbiamo tradotto come microcalcolatore, non è che una piccola parte di un "sistema microcalcolatore", la cui composizione esterna è indicata qui sotto, semplicemente come mezzo per ottenerne una prestazione.

La scheda microcomputer è situata
all'interno della "cassetta" microcomputer
che costituisce appunto un componente
del "sistema" microcomputer.
Il sistema comprende anche
un terminale video display,
il driver di "floppy disk"
ed una stampante.

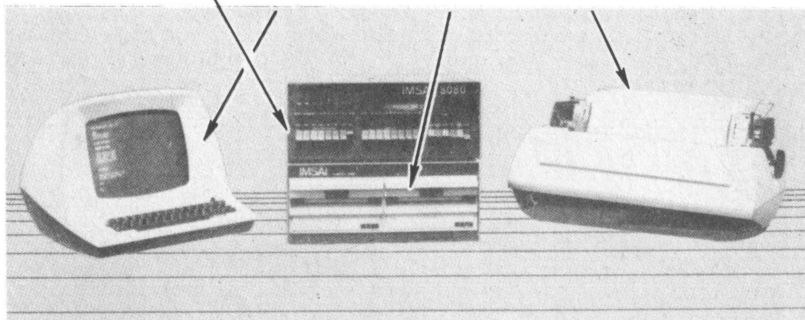



Figura 1-1. Un tipico sistema a Microcomputer

A questo livello molto superficiale, non è ancora necessario sapere che ci stiamo riferendo ad un calcolatore (e, in molti casi, è meglio!)

Se, per esempio, voi usate il sistema a microcomputer per gestire le buste paga del vostro ufficio, la macchina diventa una semplice attrezzatura commerciale: essa vi offre in qualche modo la possibilità di inserirvi dentro le informazioni riguardanti tariffe ed orari e vi fornisce dati e schede (si spera!) accuratamente redatti.

Ciò significa che, purché voi riceviate le informazioni richieste nel modo desiderato, non vi interessa (e non vi serve) conoscere come il calcolatore funziona, esattamente come non è necessario conoscere il funzionamento di un jet per volarci dentro.

Però, se è vero che non c'è valido motivo per imparare come funziona un jet (se, seduto assieme a voi, volasse il progettista di quel jet, la sua competenza non lo farebbe arrivare prima di voi), è anche vero che lo stesso discorso non si può esattamente applicare nel caso del calcolatore.

L'ignoranza del suo funzionamento significa doversi fidare della completa onestà o abilità del programmatore, o dover sbattere ogni tanto il naso sul fatto che "si tratta di un errore del computer". Ecco allora l'importanza (e ci riferiamo qui solamente alla vita di tutti i giorni, non ancora all'hobby) di imparare come funzionano i calcolatori.

Ed è proprio questo che cominciamo ad insegnarvi.

UN SISTEMA MICROCALCOLATORE

Iniziamo allora col prendere in esame un intero sistema microcalcolatore, analizzando il funzionamento di ogni singola parte.

IL DISPLAY VIDEO (O MONITOR)

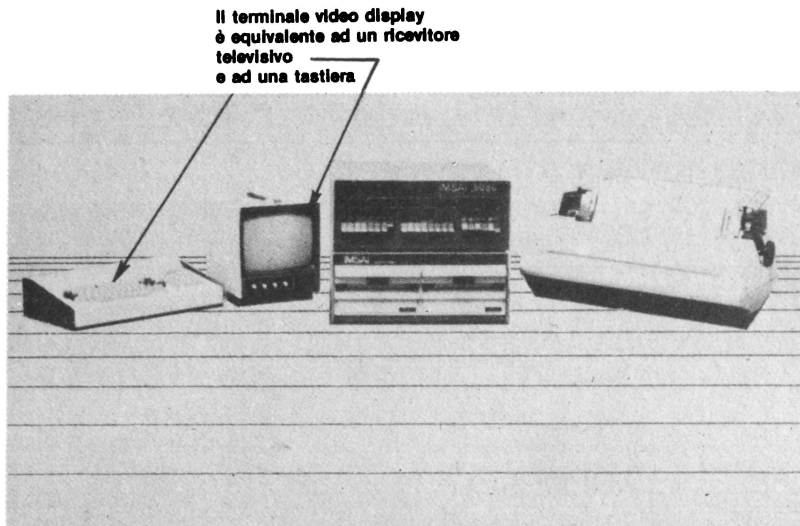
La parte per noi più importante del sistema della figura precedente è costituita dalle interfacce uomo-macchina.

Voi intrattenete, col sistema microcalcolatore, un dialogo che ha lo scopo di fornirgli i dati in entrata; **il sistema parla a voi evidenziando i messaggi appropriati sullo schermo video.** Voi rispondete battendo una risposta sulla tastiera; qualunque cosa voi abbiate battuto viene riprodotta sullo schermo, sia per controllare che non abbiate commesso alcun errore, sia per verificare che il microcalcolatore abbia correttamente acquisito i vostri dati.

Il video display, o monitor, è poco diverso da un normale televisore; la differenza fondamentale è che il video display presenta una "risoluzione" o "definizione" molto migliore, in quanto vi sono, da leggere senza troppa fatica, caratteri piuttosto piccoli.

Nel caso si accetti di lavorare con caratteri grandi, e si voglia ridurre la spesa d'impianto, si può anche utilizzare, come monitor, il normale televisore, come nell'esem-

pio dell'applicazione qui sotto rappresentata.

Il terminale video display è equivalente ad un ricevitore televisivo e ad una tastiera

Ora, uno dei problemi associati coi microcalcolatori e minicalcolatori è costituito dalla terminologia e dal linguaggio.

Mentre una buona fetta della terminologia in uso in questo settore non è altro che gergo (se non posa), tuttavia una certa parte di termini specifici è pur necessaria.

Ricordiamo che l'inglese, principale strumento di questa terminologia, si è evoluto (come del resto ogni altra lingua) in società per lungo tempo tutt'altro che tecniche, col risultato che noi spesso dobbiamo sforzarci per trovare parole, o anche frasi, che rappresentino adeguatamente dei concetti tecnici.

In questo corso vi abitueremo anche alla terminologia dei calcolatori, se non altro affinché possiate poi leggere altri testi in materia.

Rivolgendoci ora ai principianti, perchè abbiamo parlato di "video display" e non di uno schermo televisivo? La risposta, molto semplicemente, è che uno schermo televisivo è progettato per riprodurre, come scopo primario, delle immagini composite; un display video è progettato per riprodurre, come scopo primario, delle parole stampate. E vi sono differenze sostanziali fra l'elettronica che meglio si adatta a ciascuno dei due casi.

Uno schermo televisivo, in quanto deve riprodurre immagini, deve quindi poterne interpretare i neri, i bianchi e i grigi (oppure i vari colori e le loro intensità), ma non ha problemi di alta risoluzione per via di piccoli caratteri e dettagli molto fini.

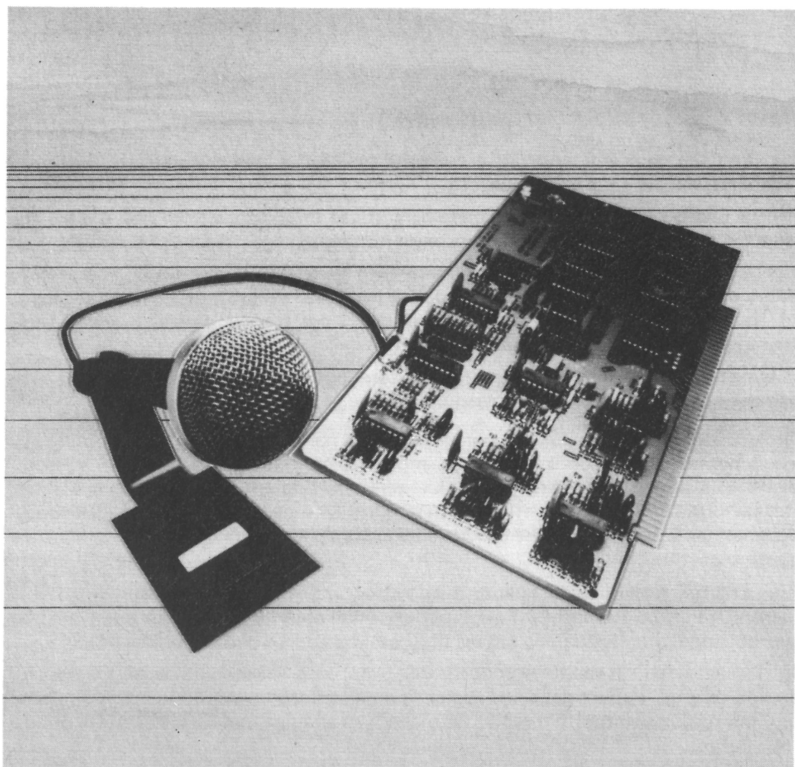
Un display video, dovendo riprodurre del testo, sarà interessato da neri e bianchi, ma non da grigi (oppure dai singoli colori, non dalle loro tonalità); dovrà però presentare una risoluzione molto elevata, dato che ha à che fare con caratteri molto minuti e ben definiti.

Naturalmente, se i costruttori di televisori adottassero tubi televisivi di alta qualità, e cioè con la risoluzione richiesta dai terminali a display video, allora il vostro televisore potrebbe servire anche come monitor per calcolatore senza alcuna perdita di qualità; ma ciò significherebbe pagare nettamente di più il televisore.

CRT
VDU

Ci si riferisce al monitor video talvolta indicandolo come tubo a raggi catodici (CRT) oppure come unità display video (VDU); le abbreviazioni indicate seguono la terminologia inglese. La dicitura più logica ci sembra "video display unit", in quanto "cathode ray tube" si riferisce al modo in cui i tubi per visualizzare segnali televisivi o messaggi scritti vengono oggi costruiti; e, in un futuro non troppo lontano, i tubi catodici saranno molto probabilmente sostituiti da display a diodi fotoemittenti, o altri dispositivi analoghi.

LA TASTIERA

Ritorniamo al nostro sistema microcalcolatore per esaminare la tastiera, che al giorno d'oggi è l'unico sistema a nostra disposizione per fornire nuovi dati ad ogni sistema calcolatore, almeno ai tipi medio-piccoli. Nel futuro sarà probabilmente possibile sostituire una tastiera direttamente con un microfono, e parlare direttamente col vostro computer.

Esistono sul mercato numerosi tipi di tastiere, ognuna delle quali basa il suo diritto ad esistere sulla scorta di particolari prestazioni o possibilità che ognuno di voi potrà trovare volta a volta fondamentali o ridicole.

INTERFACCIA

Voi però potete anche sostituire la tastiera con una telescrivente, a patto che risulti disponibile un'appropriata "interfaccia"; ma, di cosa si tratta?

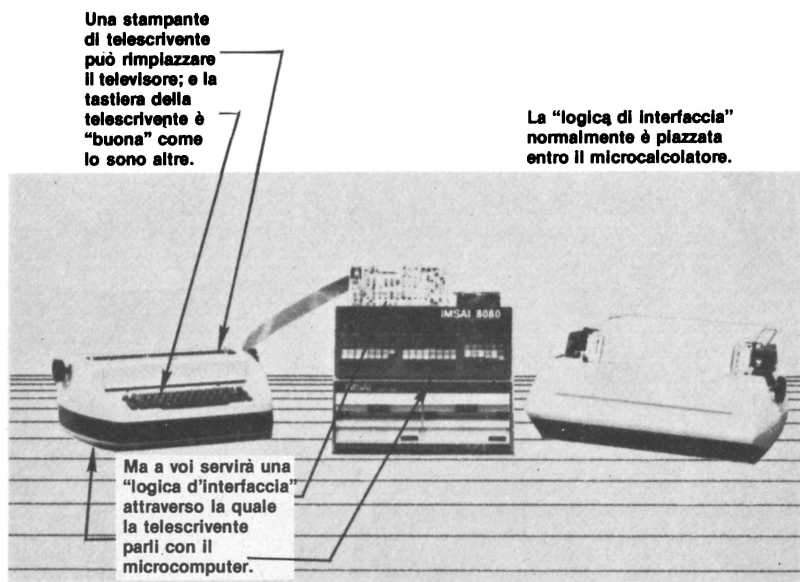
Naturalmente, se voi semplicemente piazzate la vostra telescrivente vicino al microcalcolatore ed al televisore, battendo sui tasti non si ha alcun effetto sul display video. Qualcuno dovrà fornirvi la circuiteria elettronica che sente lo spostamento dei tasti e ne crea segnali appropriati, tali che il microcalcolatore ed il televisore avvertano che un tasto è stato premuto.

Tutto ciò è semplicemente illustrato nella figura che segue.

Una stampante di telescrivente può rimpiazzare il televisore; e la tastiera della telescrivente è "buona" come lo sono altre.

La "logica di interfaccia" normalmente è piazzata entro il microcalcolatore.

IMSAI 8080

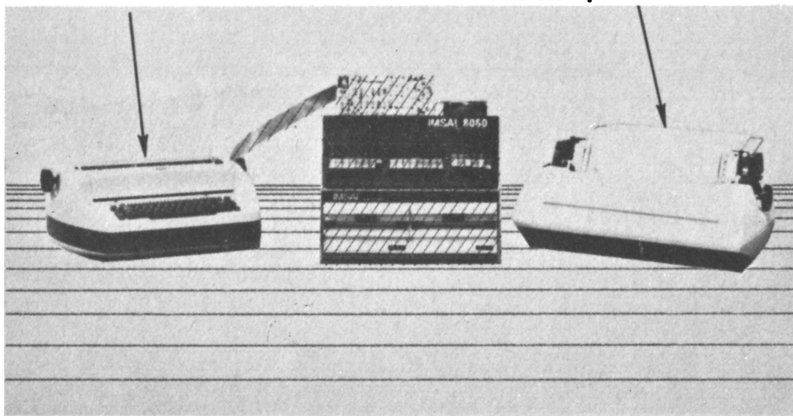
Ma a voi servirà una "logica d'interfaccia" attraverso la quale la telescrivente parli con il microcomputer.

Come potete immaginare, la tastiera, il display video, in poche parole, ciascuno dei componenti di un sistema a microcalcolatore, devono avere la propria interfaccia elettronica verso il microcalcolatore stesso.

Solo in tal modo una tastiera ed un monitor possono essere sostituiti da una telescrivente ed un televisore.

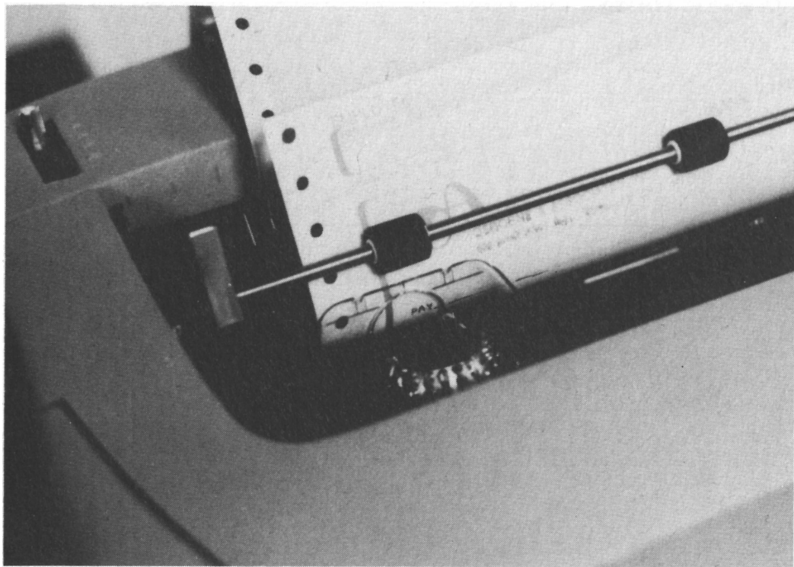
LA STAMPANTE

La telescrivente offre qualcosa che una tastiera non può fare: la telescrivente può scrivere quello che voi battete. Nel nostro sistema microcalcolatore mostriamo una stampante addetta appunto a tale funzione.

La stampante della telescrivente è sostitutiva del televisore.

È qui rappresentata una stampante separata che esegue l'attuale lavoro di stampa.

Se sostituiamo una telescrivente alla tastiera, possiamo eliminare la stampante. Ma allora, perchè la gente compra stampanti? La risposta è: per la velocità di stampa. Le telescriventi sono farraginosi apparati meccanici che non reggono a velocità superiori ai 15 caratteri al secondo. Le stampanti più lente sono due volte più veloci, le più veloci possono arrivare a stampare anche qualche migliaio di righe al minuto.

Ma, la velocità di 15 caratteri al secondo è veramente troppo lenta? Chiaramente nessuno potrebbe scrivere così velocemente; ma occorre ricordare che la stampante non è lì solo per scrivere quello che voi battete, ma anche per stampare programmi da voi predisposti automaticamente: ed è proprio in queste fasi che la stampante lavora al massimo.

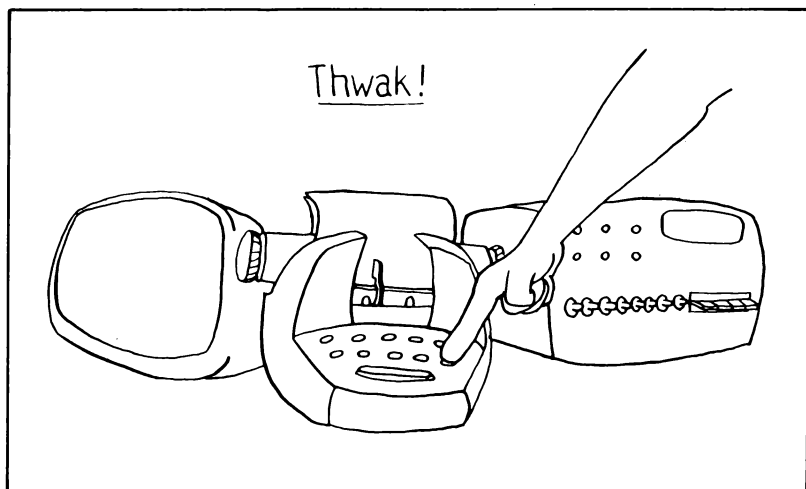
Consideriamo un programma per i fogli paga: il microcalcolatore dovrà calcolare i dati delle varie paghe, e poi stampare appunto i fogli paga, senza concorso della mano dell'uomo.

Dopo che sono stati inseriti, mediante la tastiera, tutti i dati, la prassi è quella di inserire il pacco di fogli nella telescrivente ed allineare correttamente i fogli paga.

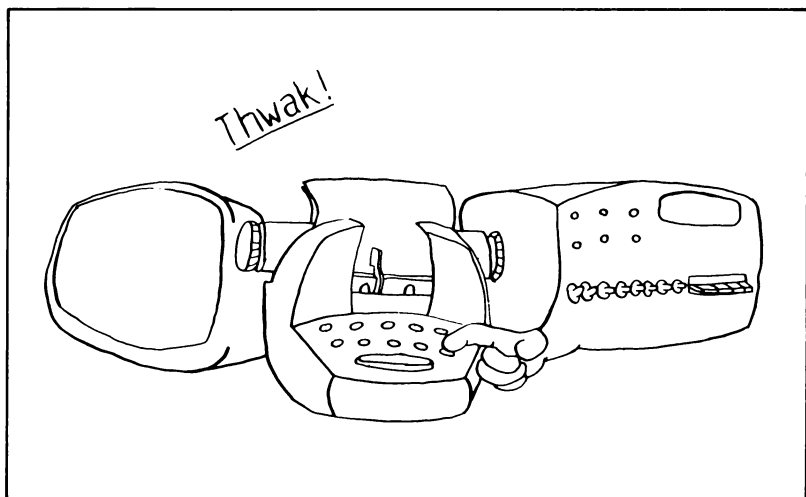
Quando il microcomputer ha terminato i suoi conti, inizia a stampare i fogli paga, mentre voi andate a prendere il caffé o fate qualcos'altro. Se voi avete un ruolino paghe di 50 dipendenti, il sistema microcalcolatore equipaggiato con una telescrivente da 15 caratteri/secondo risulterà impegnato per circa 45 minuti.

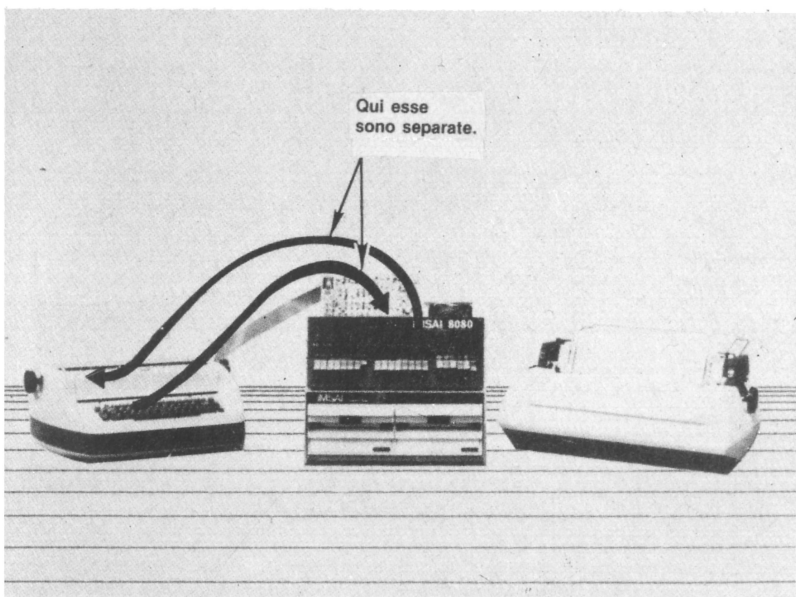
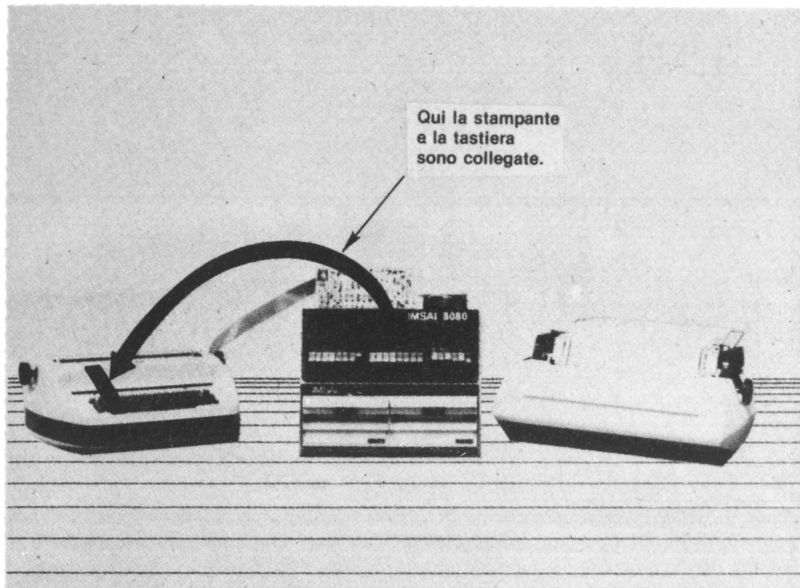
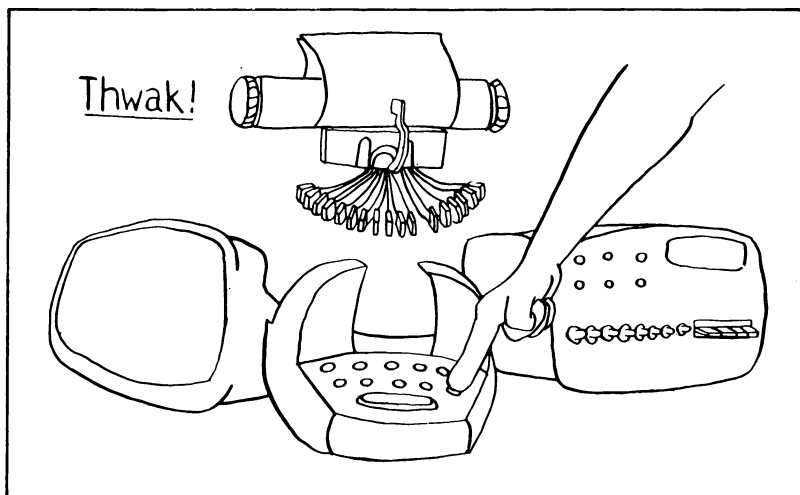
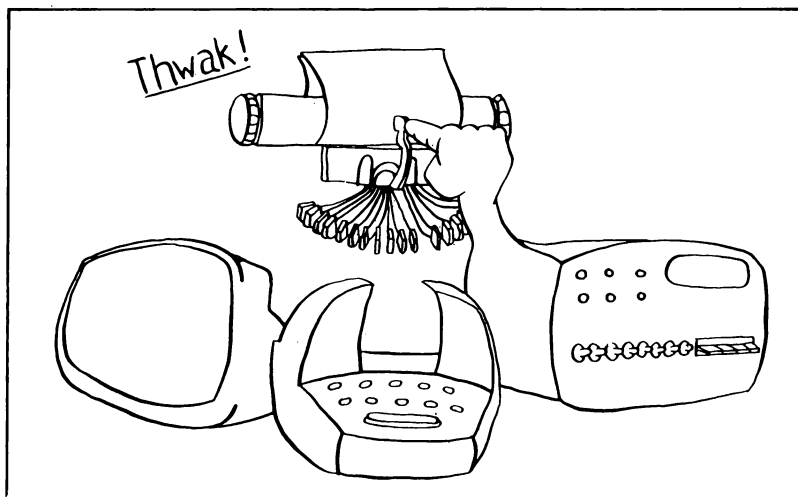
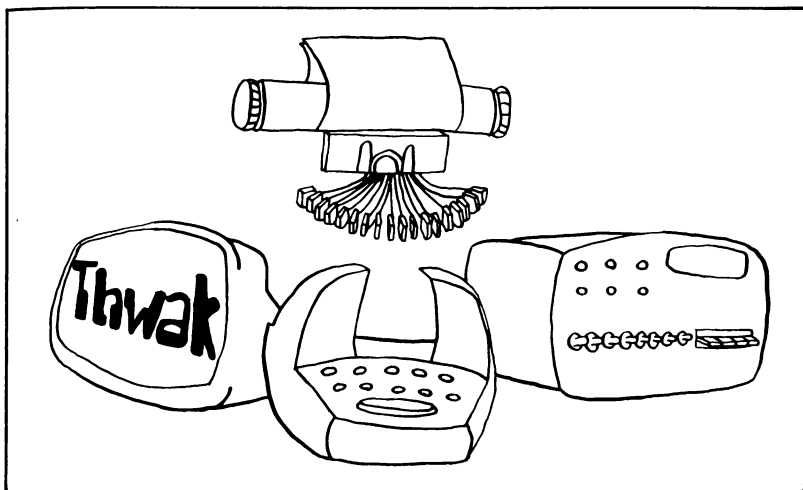
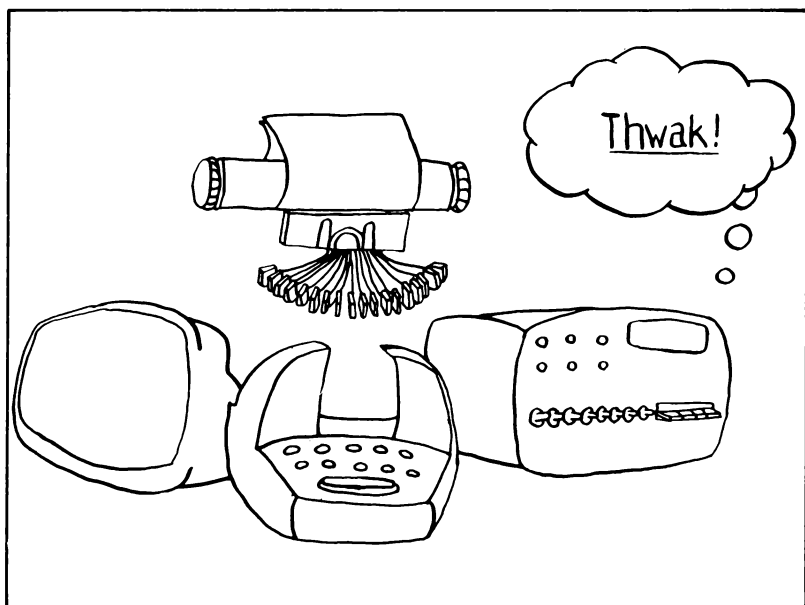
Sostituendola con una stampante veloce, voi potete enormemente ridurre il tempo di stampa, e rendere così libero il calcolatore per altri usi molto più presto.

Per esempio una stampante da 150 caratteri al secondo, ancora abbastanza normale, permetterà di completare la stampa dei suddetti fogli paga in 4 minuti e mezzo, tali quindi da far sembrare il tempo precedente una vera agonia.

Ma la velocità di stampa non è l'unica ragione per mantenere separata la stampante dalla tastiera. Quando esse sono collegate, come avviene in una telescrivente, ogni volta che voi premete un tasto, stampate un carattere.

Thwak!

Quindi, ogni volta che il microcalcolatore stampa un carattere, esso costringe un tasto ad abbassarsi.

Thwak!

Ciò significa che voi non potete usare la tastiera mentre state stampando: e viceversa, voi non potete stampare mentre state usando la tastiera.

Disaccoppiamo la stampante dalla tastiera: concettualmente accade ciò che è indicato nelle due figure qui sotto.

Quando la tastiera e la stampante sono accoppiate meccanicamente, premendo un tasto automaticamente si stampa un carattere.

Quando la stampante e la tastiera sono disconnesse, il microcalcolatore deve essere programmato appositamente per stampare il carattere battuto, altrimenti questo non verrà stampato.

Alternativamente il microcalcolatore può girare il carattere al display video.

Oppure il microcalcolatore può non mandare affatto il carattere.

ECO **Se il microcalcolatore riporta indietro un carattere, o alla stampante o al display, si dice che esso "echeggia" un carattere, cioè che ne produce l'eco.**

Noi però abbiamo appena dimostrato un concetto importante: il controllo indipendente di dispositivi apparentemente legati da parte del microcalcolatore.

Il fatto che una tastiera ed un display video possano essere collocati entro una singola apparecchiatura non implica infatti che ogni qualvolta voi abbassate un tasto venga automaticamente riprodotta una lettera.

Se l'eco si verifica, esso è provocato dal microcomputer.

Il fatto che una telescrivente fisicamente colleghi la sua tastiera col suo elemento scrivente significa invece che, ogni qualvolta voi ne premete un tasto essa vi stamperà un carattere, che voi lo vogliate o no. Infatti il microcalcolatore non ha poteri di controllo sull'eco di una telescrivente, e questo costituisce una caratteristica indesiderata di un sistema microcomputer.

COMPONENTI CHE IMMAGAZZINANO UNA QUANTITA' DI DATI

PROGRAMMI **Abbiamo già, e più volte, enunciato diverse cose che il microcalcolatore deve fare: ma, queste cose, come le fa? Il definire cosa un calcolatore deve fare ne costituisce la "programmazione".**

Più avanti andremo ad esaminare la programmazione concettualmente; nel CORSO BASE essa sarà descritta in dettaglio.

Al momento, ci interessa solo esaminare le parti individuali di un sistema microcalcolatore; perciò, piuttosto che discutere su "come" scrivere i programmi, esaminiamo "di cosa avrete bisogno" per scriverli.

Ciascun programma di calcolatore consiste semplicemente in una sequenza di numeri.

Questi numeri non hanno assolutamente niente di speciale, ma sono puri e semplici numeri; **semplicemente, un numero immagazzinato entro un microcalcolatore può rappresentare un passo o un elemento del programma, oppure può rappresentare un "dato".**

E' tutta questione di interpretazione: se il microcomputer acquisisce un numero quando gli serve un passo elementare di programma, allora esso semplicemente prende per buono che il numero acquisito è un passo del programma.

Ma se il microcalcolatore si aspetta di ricevere un numero (per esempio, uno dei due numeri che devono essere sommati) allora esso capisce che il numero in arrivo è un "dato"

DATI **"Dato" è una parola usata per descrivere numeri quando essi devono essere interpretati come numeri o lettere dell'alfabeto, in contrapposizione ai numeri che devono essere interpretati come passi del programma.**

Per esempio, se si sommano due numeri, questi due numeri, più la risposta che si genera (cioè la somma), costituiscono dei "dati".

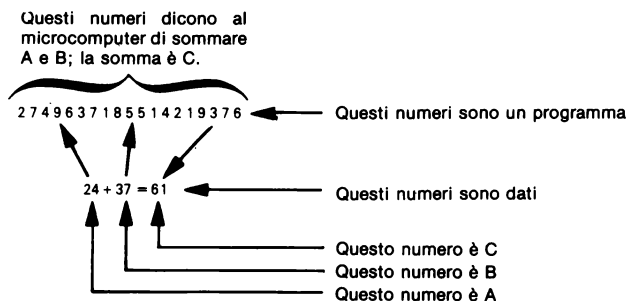
Le istruzioni al microcalcolatore, quelle che gli fanno sommare i due numeri, costituiscono un programma: il programma stesso consiste di una sequenza di numeri, ma questi numeri, ora non rappresentano dei dati, bensì dei passi di programma.

Ciò può essere illustrato dal grafico.

Questi numeri dicono al microcomputer di sommare A e B; la somma è C.

2 7 4 9 6 3 7 1 8 5 5 1 4 2 1 9 3 7 6 ← **Questi numeri sono un programma**

24 + 37 = 61 ← **Questi numeri sono dati**

Questo numero è C
Questo numero è B
Questo numero è A

Non c'è niente di anormale in numeri che, entro un microcalcolatore, rappresentino sia passi di programma che dati veri e propri: noi tutti facciamo qualcosa del genere tutti i giorni. Per esempio, un numero su un pezzo di carta può essere parte del numero di matricola di una polizza di assicurazione; potrebbe anche essere il numero di controllo di un assegno, come il suo ammontare in lire, o la cifra di un bonifico.

E' solo mediante il controllo e l'interpretazione che voi potrete dire di cosa si tratta.

Se, mentre state registrando la distinta della vostra banca, accidentalmente vi capita di leggere il numero della distinta invece della somma dell'accredito, avrete effettuato una registrazione bancaria molto attraente (e ovviamente sbagliata); ma non c'è niente di obiettivamente impossibile nel commettere questo errore. Allo stesso modo, potete capire come non vi sia niente di inerentemente impossibile nel fatto che un microcomputer legga un numero che è un dato e lo interpreti invece come passo di programma; i risultati sarebbero comunque molto strani.

I numeri rappresentano appena una piccola parte delle nostre parole; essi invece costituiscono una parola completa per i microcalcolatori.

Anche un piccolo compito, quando è stato elaborato come programma per un microcalcolatore, darà origine a molte centinaia di numeri. Dal momento in cui voi avete definito tutti i compiti che volete che il microcomputer metta in pratica, tutti questi programmi così elaborati potranno diventare molte migliaia, o anche milioni, di numeri. E ciò, naturalmente, pone dei problemi.

LA MEMORIA

MICROSECONDO

Il microcalcolatore realizza ogni compito specifico affidatogli eseguendo un programma specifico. Il programma consiste in una sequenza di passi, o istruzioni; ogni istruzione è identificata da un unico numero.
Ma un microcalcolatore può passare attraverso centinaia di migliaia di passi di programma, o istruzioni, in un solo secondo.

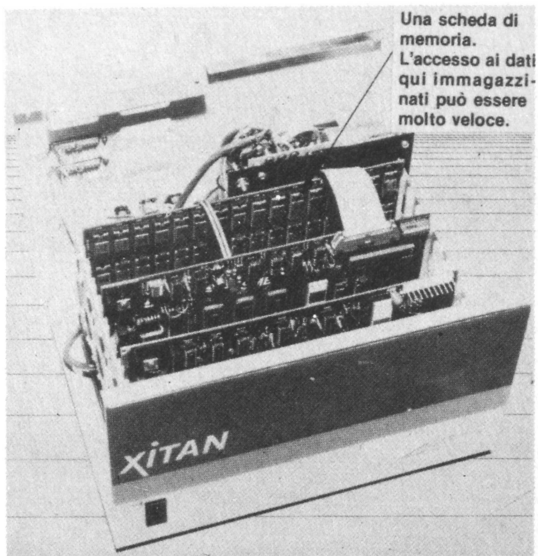
Infatti, il tipico microcalcolatore eseguirà una singola istruzione in un qualche intervallo di tempo che si può aggirare da 1 a 10 milionesimi di secondo (con maggior pertinenza, li chiameremo microsecondi).
Se il microcalcolatore è tale da eseguire un'istruzione ogni qualche microsecondo, è allora chiaro che il numero rappresentativo dell'istruzione dovrà essere immagazzinato in qualche tipo di "magazzino" ad accesso veloce e di pronta risposta.

Il microcomputer deve essere in grado di "operare" il numero che rappresenta l'istruzione in un tempo perfino inferiore ai pochi microsecondi disponibili per eseguire l'intera istruzione.

Un microcalcolatore sarà in grado di estrarre un numero di istruzione da un "magazzino" ad accesso rapido in tempo tipico di mezzo microsecondo, o anche meno.

Però questo tipo di "magazzino" a pronta risposta ed accesso veloce è costoso.

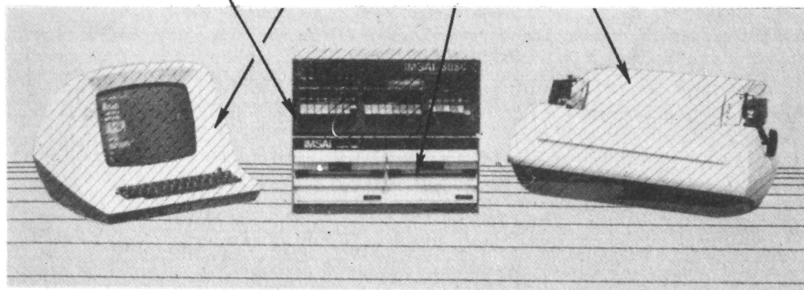
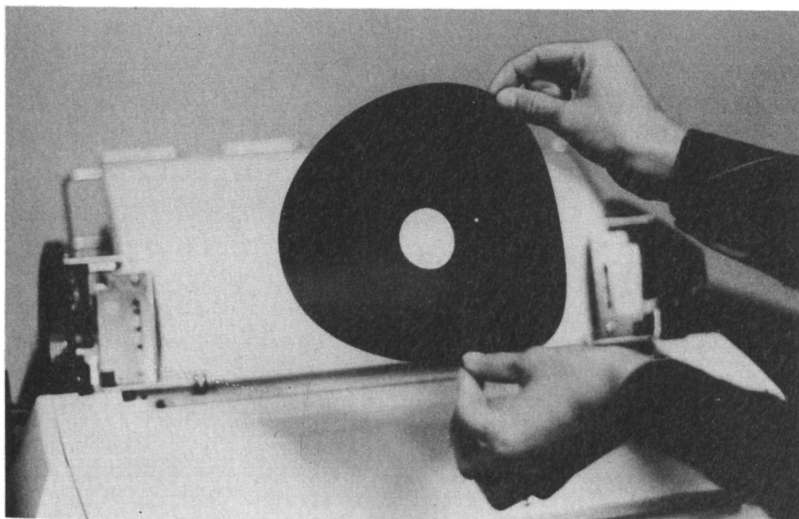
Il microcomputer sarà quindi equipaggiato da una quantità relativamente piccola di "magazzini" ad accesso veloce, che chiamiamo memoria.

Una scheda di memoria. L'accesso ai dati qui immagazzinati può essere molto veloce.

Questa memoria ad accesso veloce è normalmente "nascosta" dentro la scatola del microcomputer.

I passi ed i dati di programma che il microcalcolatore usa correntemente dovranno essere immagazzinati in questa memoria ad accesso veloce.

UNITA' A "FLOPPY DISK"

I programmi ed i dati che il microcalcolatore non usa correntemente vengono immagazzinati usando alcuni tipi di grossi dispositivi di immagazzinamento più lenti (ed anche economici), come nella figura che segue.

La scheda microcomputer è dentro l'involucro del microcomputer vero e proprio, che a sua volta è solo un componente di tutto il sistema microcalcolatore. Tale sistema possiede anche un terminale a display video, una stampante ed è alimentato da floppy disk.

Esistono similitudini concettuali fra un sistema a "floppy disk" ed un giradischi. Tale sistema immagazzina il suo corredo di informazioni appunto su "floppy disk", che sono così chiamati in quanto si tratta di dischi soffici e facilmente pieghevoli (da flop, o flap: cadente, floscio).

Poiché questi dischi sono "floppy", essi sono contenuti in astucci di cartone, appunto per mantenerli rigidi.

MINI FLOPPY

Esistono due misure di "floppy disk": quelli normali, che hanno 20 cm di diametro, e i "mini floppy", che hanno circa 13 cm di diametro.

Mentre un disco normale ha una superficie a solchi, **il "floppy disk" ha la superficie liscia magnetica; su tale superficie, l'informazione viene immagazzinata sotto forma di sequenza, di impulsi magnetici.**

PISTE

Gli impulsi magnetici sono registrati lungo delle "piste" sulla superficie del "floppy disk". E ciò in contrasto coi dischi normali, nei quali la musica è immagazzinata sulla superficie mediante un solco continuo, che viene seguito ed analizzato da una puntina.

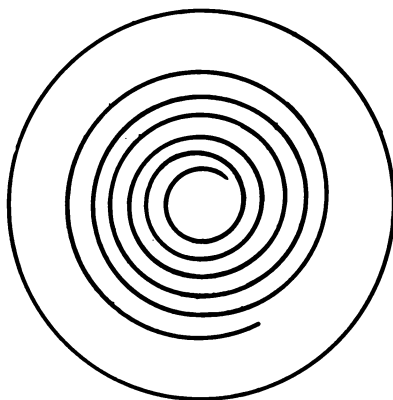
Poiché invece la superficie del disco è completamente liscia, senza cioè alcun solco inciso, la pista diventa la linea immaginaria lungo la quale sono posizionati i diversi impulsi magnetici.

L'informazione viene scritta sulla pista, e viene letta, da una testina magnetica legge/scrive, che è molto simile alla testina di un giradischi; naturalmente, la testina del "floppy disk" non ha la puntina, bensì un piccolo tampone metallico che può generare (per scrivere) o rivelare (per leggere) gli impulsi magnetici sulla superficie del "floppy disk".

Ora, finchè non sarete diventati veramente degli esperti in microcalcolatori, non è necessario intrattenersi a lungo su come esattamente l'informazione viene immagazzinata nel disco. Perciò la discussione che segue vi permetterà una comprensione generale dei concetti, e niente di più.

La maggior parte degli utenti di microcomputer non si disturba ad acquisire queste informazioni, esattamente come buona parte degli amanti della musica non si preoccupa di studiare come tale musica venga registrata su nastri o dischi.

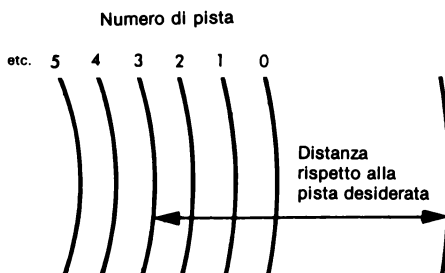
Quello che importa sapere è che un disco, quando suonato, genera musica, un "floppy disk" genera numeri: gli stessi numeri che vi sono stati scritti sopra, esattamente come la musica che è stata registrata.

Sarebbe possibile registrare informazioni sulla superficie di un floppy, più o meno come la musica è registrata sulla superficie di un disco, semplicemente disponendo di un solco continuo (o meglio, nel nostro caso, di una pista continua) che si svolge a spirale partendo dal centro della superficie del disco.

Il problema, con questo sistema di registrazione, risiede nella difficoltà notevole di azzeccare il punto giusto dell'informazione registrata.

Potete verificare ciò da voi stessi, tentando di trovare una particolare parola in una canzone su un disco.

Se lottate abbastanza a lungo, probabilmente riuscirete nell'impresa; ma otterrete ciò solo basandovi sul vostro giudizio umano.

L'elettronica non può usare il giudizio umano; può solo seguire regole fisse.

E' per questo che sostituiamo la singola spirale continua con un gran numero di piste concentriche.

Ora possiamo scegliere una pista particolare mediante un suo numero di pista — che può essere indirizzato in termini di distanza (esattissima) dal bordo o dal centro del floppy.

Costruttori diversi usano numeri diversi di piste su ciascuna superficie del floppy disk. L'unico standard tuttora esistente è per i floppy da 8" (20 cm) che molto di frequente portano 77 piste concentriche (numerate da 0 a 76) registrate su una superficie.

Alcuni tipi di floppy hanno dati immagazzinati su una sola superficie del disco, mentre altri hanno dati su ambedue i lati.

Sulla superficie di un floppy da 8" si possono immagazzinare circa 250.000 caratteri di informazione; ma se voi doveste semplicemente dividere quest'informazione sulla base del numero delle piste, avreste qualche problema. Prima di tutto, le piste aumentano di lunghezza allontanandosi dal centro della superficie del disco verso il bordo, e ciò significa che non esistono due piste che immagazzinano la stessa quantità di informazione.

In secondo luogo, la quantità di informazione immagazzinata nella pista potrebbe essere molto ampia; ma essa rappresenterebbe ancora la più piccola unità singola di informazione indirizzabile sulla superficie di un floppy disk.

Se si individuasse la superficie di un tale disco solo per mezzo del numero di pista, allora si dovrebbe leggere l'informazione da un'intera pista, o scrivere l'informazione su un'intera pista.

SETTORE

Possiamo risolvere questi due problemi dividendo a settori ogni pista, ed immagazzinando la stessa quantità d'informazione su ciascuna pista, indipendentemente da quanto la pista è vicina al centro od alla periferia.

Com'è illustrato nella figura, ciò significa che una parte sempre maggiore della pista viene sprecata come ci si sposta dal centro verso la periferia del disco.

Ora voi potete identificare l'informazione sulla superficie del floppy sia dal numero di pista che dal numero del settore all'interno della pista stessa. In figura sono indicati 26 settori (numerati da 1 a 26), in quanto questo è un numero comunemente usato.

Normalmente, entro ogni settore sono immagazzinati 128 caratteri di informazione; si può quindi calcolare la capacità totale di immagazzinamento di un singolo floppy come segue:

$$\underbrace{128}_{\text{caratteri per settore}} \times \underbrace{77}_{\text{piste}} \times \underbrace{26}_{\text{settori per pista}} = \underbrace{256.256}_{\text{caratteri}}$$

FLOPPY DISK A DOPPIA DENSITÀ

Alcuni costruttori introducono 256 caratteri d'informazione in ogni settore di pista; da qui la definizione di dischi a densità doppia. Tali dischi presentano cioè la stessa dimensione di settori, ma una quantità di informazione più condensata.

Settori

Piste

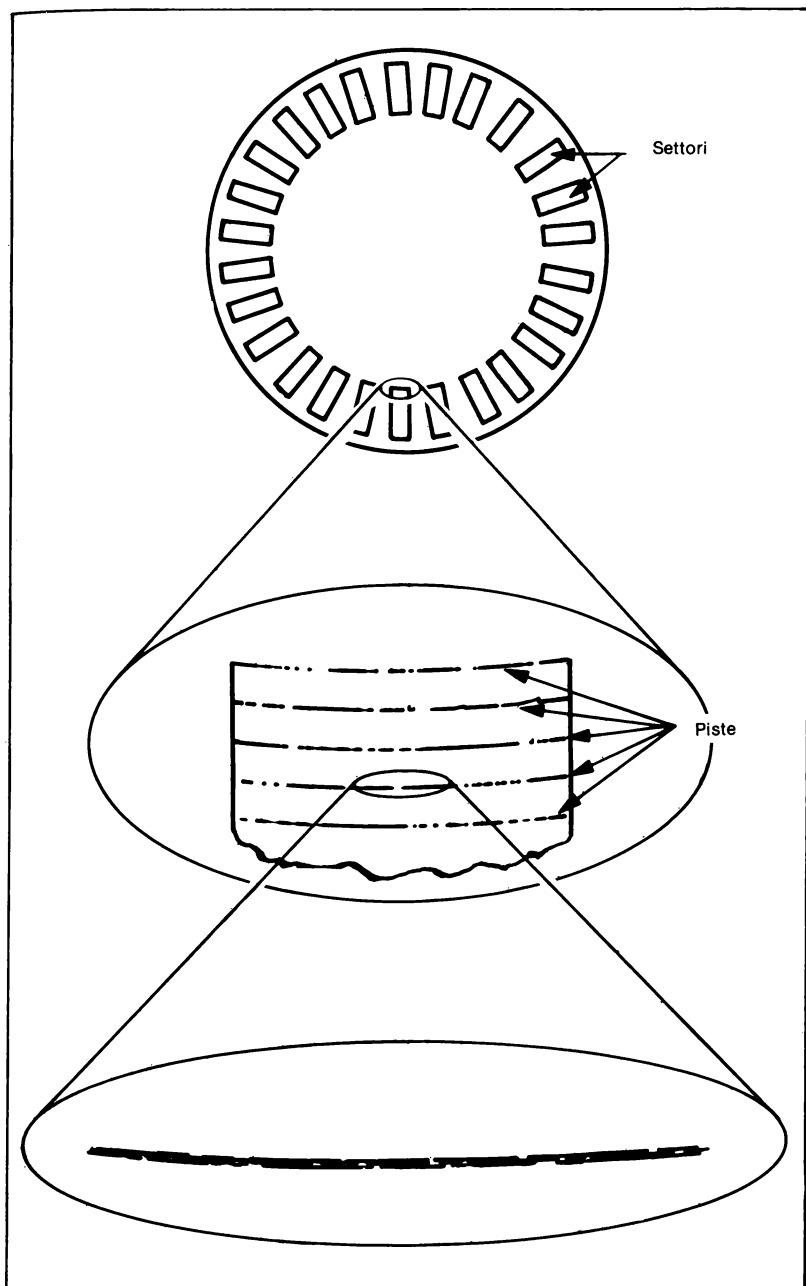
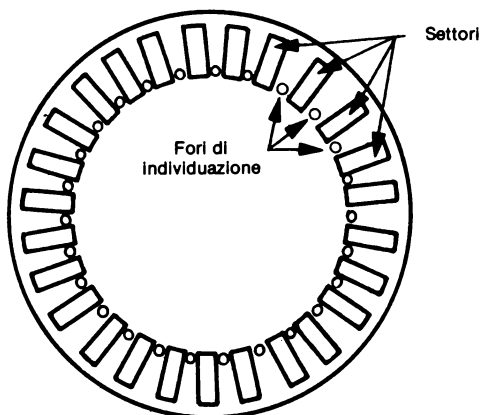
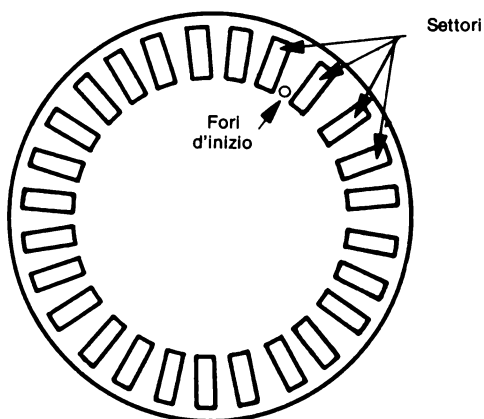



Figura 1-2. Superficie registrata di un floppy-disk

DISCHI A SETTORIZZAZIONE "DURA"

I floppy disk possono avere uno o più fori sulla superficie per aiutare il meccanismo di guida a selezionare i settori. Alcuni dischi hanno un foro punzonato fra ciascuna coppia di settori; essi sono indicati come dischi duri, e corrispondono alla figura qui sotto.

DISCHI A SETTORIZZAZIONE "MORBIDA"

I dischi che hanno un foro solo sono indicati nell'altra fifigura.

FORMAT

Quando voi acquistate un disco "bianco" cioè vergine (tale in quanto la sua superficie è completamente vuota di informazioni), in tale situazione non potete scrivere alcunché.

Infatti, prima di poter scrivere su un floppy disk vergine, occorre passare attraverso una fase preliminare detta di "formatting".

Durante tale fase sul disco vengono tracciati i settori e le piste, usando opportuni codici magnetici; si tratta di un procedimento automatico che voi realizzate, usando il vostro microcalcolatore, semplicemente seguendone le istruzioni; si tratta comunque di una fase indispensabile.

Una volta che il vostro disco è stato "formattato", esso è pronto per l'uso; vale a dire che potete registrarvi sopra l'informazione oppure leggerla.

In conclusione, il bello dei floppy disk è che voi potete averne una collezione, esattamente come potete avere una collezione di registrazioni musicali. Potete quindi acquistare floppy disk già completi di programma, o comprarli vergini e registrarvi da soli i vostri programmi, in modo che, in ogni momento, potrete estrarre il vostro floppy dalla libreria ed inserirne il programma.

UNITA' A DISCHI RIGIDI

I floppy disk non sono il solo mezzo per immagazzinare dati e programmi; infatti i grossi sistemi calcolatori usano grandi dischi rigidi.

I sistemi a grandi dischi rigidi offrono una enorme varietà di opzioni e dimensioni; in ogni caso essi contengono una gran mole di informazione in più dei floppy (ma, naturalmente, sono molto più costosi).

In realtà, i dischi rigidi sono più economici se guardate al loro costo in termine di prezzo per unità di informazione contenuta.

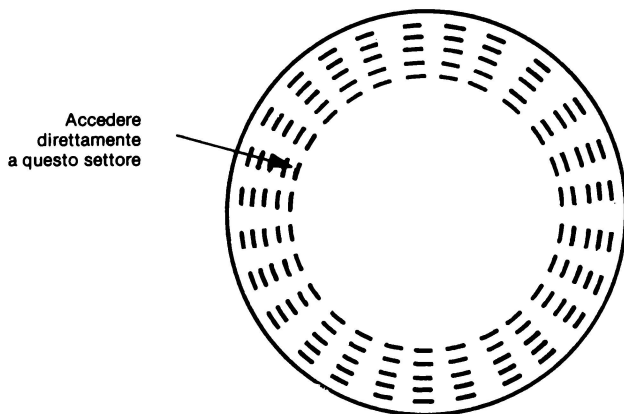
Il disco rigido è l'equivalente dei floppy disk dei sistemi più piccoli; su di esso sono immagazzinati i dati.

Ma con un piccolo sistema microcalcolatore voi non avete certo bisogno della capacità di immagazzinamento di un'unità a grande disco rigido, che inoltre potrebbe trasferire informazioni a velocità maggiore di quella alla quale il microcomputer può operare.

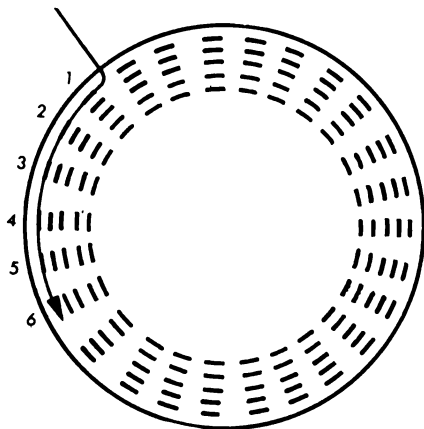
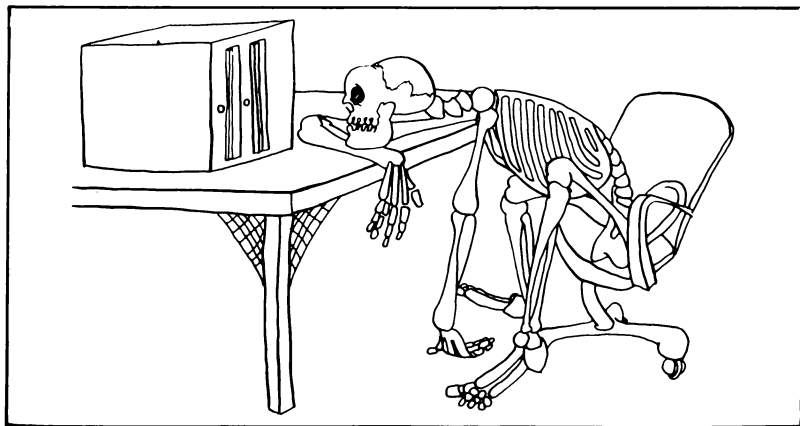
ACCESSO AI DISCHI

ACCESSO CASUALE

Sia i dischi floppy che quelli rigidi vengono indicati come grossi dispositivi di memoria ad "accesso casuale". Tale definizione significa che voi potete accedere direttamente a qualsiasi settore voi vogliate su ciascuna pista allo scopo di leggere o scrivere informazioni.

Accedere
direttamente
a questo settore

Supponiamo, per esempio, che voi desideriate leggere il contenuto del settore 8, pista 12; la testina di lettura sul floppy disk può andare direttamente a tale settore senza prima dover percorrere le piste che precedono, dalla 1 alla 11 ed i settori, dall'1 al 7, della pista 12.

L'accesso casuale è una prestazione estremamente utile in ogni grosso dispositivo di memoria. La possibilità di accedere direttamente ad ogni settore allo scopo di leggere o scrivere permette di accelerare le operazioni di accesso dati.

Supponete di dover leggere dei settori in sequenza; non avrete certo molti problemi con i primissimi settori.

Ma questo comporterebbe un tempo penosamente lungo per raggiungere uno degli ultimi settori sulla superficie del floppy.

SETTORI IN CATENA

La possibilità di accedere ad ogni settore della superficie del floppy presenta un altro vantaggio, anche se meno ovvio. Cosa succede se avete un blocco di informazione che è troppo grande per essere inserito in un singolo settore?

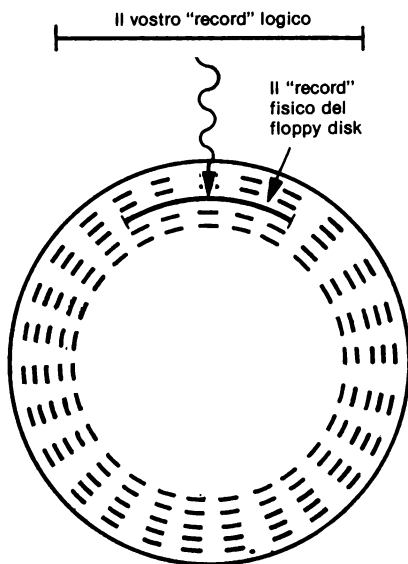
Supponiamo, per esempio, che il vostro blocco d'informazione sia così grande da dover essere immagazzinato in ben 5 settori.

In pratica, voi potreste usare il sistema microcalcolatore per scrivere lettere "tipo"; potreste per esempio immagazzinare 50 frasi standard sul floppy, e quindi creare una varietà di lettere semplicemente legando assieme frasi opportunamente scelte.

Ogni frase allora potrebbe diventare un'unità d'informazione, che nell'esempio da cui siamo partiti, è immagazzinata in 5 settori del floppy disk.

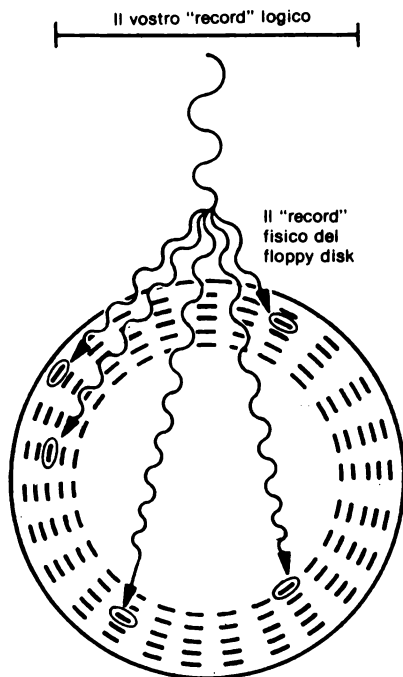
Ora, la vostra reazione più ovvia e immediata può essere quella di usare 5 settori contigui.

Il vostro "record" logico

Il "record" fisico del floppy disk

Ma, poiché voi potete accedere ai settori a piacere, non fa alcuna differenza se i 5 settori sono contigui, come illustrato qui sopra, o se essi sono sparpagliati sulla super-

ficie del disco, come qui sotto.

Il vostro "record" logico

Il "record" fisico del floppy disk

Quando voi immagazzinate una singola unità d'informazione su più di un settore, i settori si dicono "in catena".

REGISTRAZIONI LOGICHE E FISICHE

REGISTRAZIONE O "RECORD"

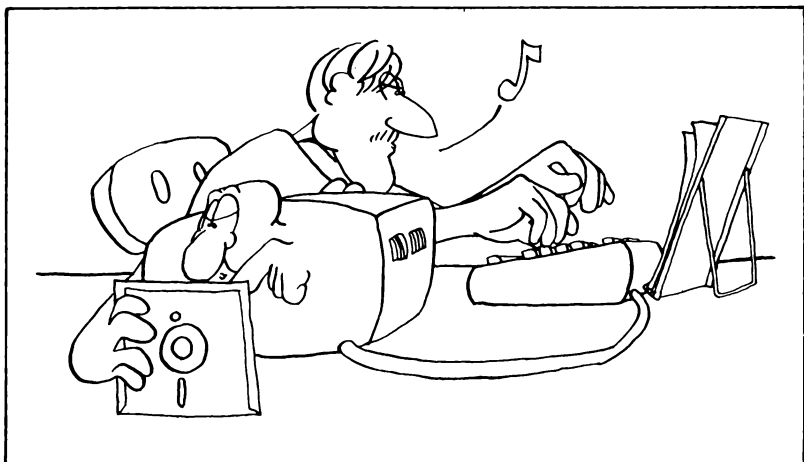
Questo è il momento giusto per introdurvi ad un concetto veramente fondamentale per i microcalcolatori: la relazione esistente fra le idee "logiche" e la realtà "fisica".

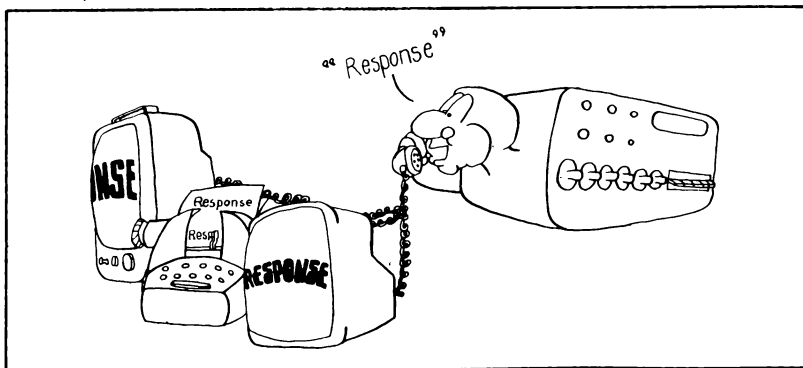
REGISTRAZIONI FISICHE E LOGICHE

Consideriamo uno dei paragrafi che fanno parte di una lettera tipo; questa frase è scritta su 5 settori. Riferiamoci a questa frase, a questo paragrafo, come ad una (unità di) registrazione". Sulla superficie del floppy disk, questa registrazione diventa una "registrazione fisica" consistente in 5 settori in catena che possono essere o non essere contigui.

Ma perché dovreste preoccuparvi dei settori? Vi renderete la vita molto più semplice se manipolerete l'informazione come insieme di "frasi" o come sarete in grado di leggerla. In realtà a voi interessa ottenere l'informazione come "registrazioni logiche" che (si spera) non hanno niente a che fare con settori o piste. E ciò è quanto una ben progettata unità floppy vi lascerà pensare.

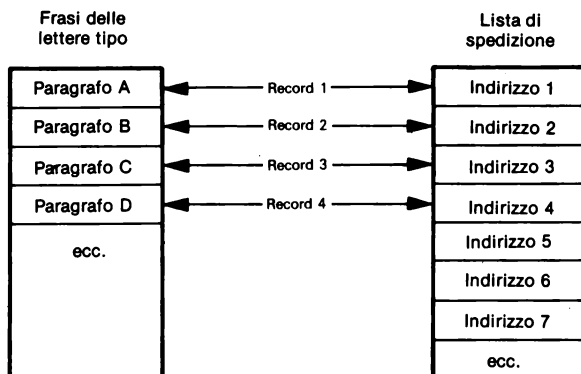
Il floppy disk si prende cura di cercare i settori e di metterli in catena, se una registrazione è contenuta in più di un settore. A voi non interessa conoscere quanti settori sono richiesti per la vostra registrazione, o dove sono dislocati tali settori, volendo, neppure sareste tenuti a sapere che esistono piste e settori.

Voi avete a che fare con "l'idea logica" della frase, ed è il microcalcolatore che si prende cura della "realtà fisica" di settori e piste.

Cossiché, quando state usando un ben progettato sistema microcomputer, potete pensare in termini logici, ignorando settori, tracce e complicazioni similari.
Però voi dovreste sempre rammentare che, proprio perché esistono settori, tracce ed accessi casuali, l'unità floppy disk è un efficace, veloce e grosso dispositivo in grado di elaborare l'informazione.
Il concetto di "logico" in confronto a "fisico" può essere esteso a coprire argomenti diversi da informazione e registrazione.
"Unità logiche" e "unità fisiche" si applicano quasi in ogni parte di ciascun sistema calcolatore.
Quella che viene generalmente definita una "unità logica" consiste in un frammento di informazione, in un'idea, o in un'operazione quale voi, essere umani, la userete.
Un'"unità fisica" è il reale completamento fisico - la realtà fisica oltre l'idea, l'informazione o la funzione.
Per esempio, nella nostra precedente esposizione, abbiamo visto come un display video può essere sostituito sia da un televisore sia da una telescrivente.

In questo caso tutti e tre i dispositivi — video display, televisore e telescrivente — sono la realtà, cioè unità fisiche che rappresentano l'idea dell'unità logica: risposta del computer.

RECORD E FILE

Supponete di avere un certo numero di frasi di lettere tipo, ognuna delle quali sia registrata come unità logica sulla superficie di un floppy disk. Considerate un'altra alternativa: una lista di nomi e indirizzi. Ciascun nome e indirizzo potrebbero essere immagazzinati come singole registrazioni logiche.

Questo è il modo in cui le due serie di registrazioni logiche potrebbero essere paragonate:

Frasi delle lettere tipo		Lista di spedizione
Paragrafo A	◄— Record 1 —►	Indirizzo 1
Paragrafo B	◄— Record 2 —►	Indirizzo 2
Paragrafo C	◄— Record 3 —►	Indirizzo 3
Paragrafo D	◄— Record 4 —►	Indirizzo 4
ecc.		Indirizzo 5
		Indirizzo 6
		Indirizzo 7
		ecc.

Allo scopo di identificare un particolare brano di informazione, dovreste ora conoscere il numero di registrazione logica e sapere se si tratta di uno dei "record" logici delle lettere tipo, o uno di quelli della lista di spedizione.

Come farete ad identificare le registrazioni logiche delle lettere tipo da quelle della lista postale?
Chiaramente non ci serve partire con lo specificare dove è stata immagazzinata l'informazione sul floppy disk; tutto lo scopo di arrivare a "record" logici nella prima fase era quello di evitare di doversi preoccupare della superficie del floppy disk. Noi quindi combiniamo tutti i "record" in un "file", che consiste semplicemente in una lista di uno o più record. Ancora una volta, voi avete a che fare con "file" logici e lasciate al microcalcolatore il compito di determinare ove il "file" fisico effettivamente sia dislocato sulla superficie del floppy. **Ora avete un "file" logico di lettera tipo, dove ogni frase è una registrazione logica, ed un "file" logico di lista di spedizione, dove ogni nome e indirizzo diventano pure un record logico.**

"File" e "record" rappresentano la struttura fondamentale usata per immagazzinare gran massa d'informazione in sistemi a microcomputer, dal più piccolo al più grosso.

Questo concetto di "file" logico é "record" logici non è molto diverso, dalla vita quotidiana di ufficio. Per esempio, supponete di dover cercare una lettera che avete ricevuto dalla ditta XYZ, che contiene quotazioni e prezzi. Voi potreste chiedere ad una segretaria di prendere la 27ª lettera della terza fila dell'armadio/schedario; oppure, più probabilmente, chiederete alla vostra segretaria di andare all'archivio della ditta XYZ e di recuperare la lettera con i prezzi. Lo schedario della ditta XYZ è l'equivalente di un "file" logico; ogni lettera contenutavi è l'equivalente di un "record" logico. La vostra segreteria è l'equivalente dell'intelligenza del micro-computer che è capace di accedere ad uno specifico elemento di informazione.

UNITA' A CASSETTE

Se voi siete un tipico utente di microcomputer, troverete (almeno inizialmente) che non siete in grado di affrontare il peso economico di un sistema a floppy disk. Allora, quali sono le opzioni meno costose?

Vi sono le cassette e c'è il nastro di carta.

Le cassette usate per immagazzinare l'informazione per un microcomputer sono esattamente le stesse che usate in casa per far musica; potete infatti acquistare un registratore al supermercato ed usarlo per rimpinzarlo di informazioni ad uso del vostro sistema microcalcolatore.

Naturalmente, è augurabile che voi usiate un registratore di alta qualità e cassette con nastri di altissima qualità, perché il sistema microcalcolatore non tollererà errori.

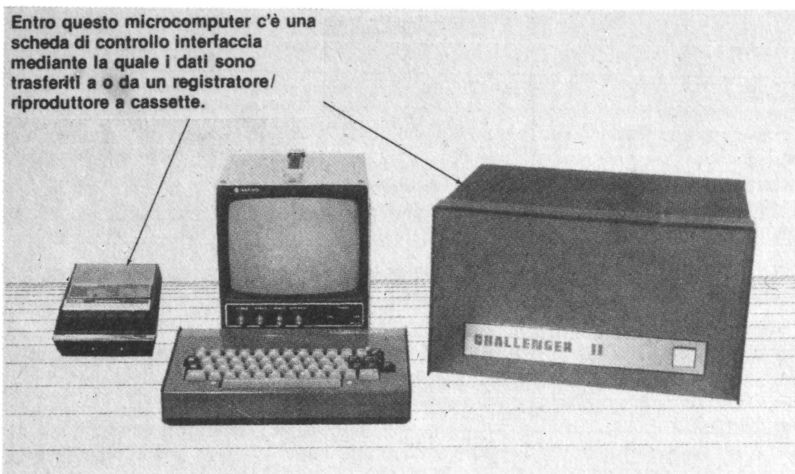
Se vi capita infatti un disturbo in una cassetta che porta musica registrata, il tutto si limiterà ad un momento di irritazione quando lo state riproducendo, ed è tutto.

Se questo disturbo si presenta in una cassetta che contiene informazioni per il microcomputer, il fatto può rendere inusabile l'intera cassetta, poiché il microcalcolatore leggerà il disturbo come una zona senza numeri o con numeri sbagliati.

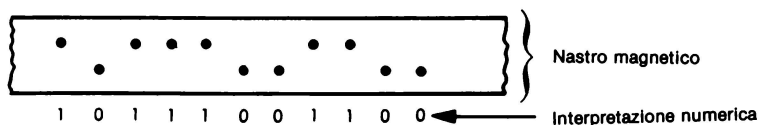
INTERFACCIA PER CASSETTE

Naturalmente, non è che voi possiate semplicemente comprare un registratore a cassette, piazzarlo vicino al vostro microcomputer ed aspettare che i due dialoghino fra di loro. La vera conversazione si verifica per mezzo di appropriate logiche di controllo all'interno del microcomputer, e la cosa è illustrata qui sotto.

Entro questo microcomputer c'è una scheda di controllo interfaccia mediante la quale i dati sono trasferiti a o da un registratore/riproduttore a cassette.

CHALLENGER II

Esistono due modi nettamente diversi in cui i registratori a cassette immagazzinano dati su nastro magnetico.
Il sistema più antico consiste nello stivare informazioni in modo digitale, vale a dire come sequenza di punti magnetizzati sulla superficie del nastro.

Nastro magnetico

1 0 1 1 1 0 0 1 1 0 0 ← Interpretazione numerica

Quasi tutti i registratori industriali a cassetta immagazzinano informazioni in modo digitale, come ora spiegato.

Invece i nuovi registratori a cassetta ora in vendita registrano l'informazione sotto forma di suoni, esattamente nello stesso modo in cui registrerebbero voci o altri rumori. Un tono particolare rappresenta il digit 0, mentre un'altra nota rappresenta il digit 1.

Il fatto che esista un "vecchio" ed un "nuovo" modo di registrare dati nelle cassette non significa che il "nuovo" sia migliore, o abbia superato il "vecchio", si tratta semplicemente di un riferimento cronologico.

Infatti, il vecchio sistema permette di stivare nella cassetta molti dati in più, permette di leggere e scrivere tali dati molto più velocemente, ma richiede nastri molto più costosi.

Perciò il vecchio sistema è migliore, ma più costoso del nuovo.

Quest'ultimo, quello cioè che usa note audio per registrare dati, ha il vantaggio principale nel permettere l'uso di qualunque registratore a cassetta di tipo domestico, sia per registratore o riprodurre la vostra musica che per costituire parte del vostro sistema minicomputer.

FLOPPY ROM

La floppy ROM è una nuova soluzione che si basa sull'uso di note per registrare dati su cassette; si tratta di una registrazione da riprodurre sul vostro giradischi.

Naturalmente, invece di ascoltarla, trasferirete l'informazione proveniente dalla floppy ROM sul nastro di una cassetta, i toni registratori sulla quale vengono interpretati come dati binari entro il vostro microcomputer.

Questi dati binari possono costituire sia istruzioni di programma, sia veri e propri dati usati per il programma, sia le due cose assieme.

Rom sono le iniziali di Read Only Memory

La registrazione contiene dell'informazione, perciò fa parte della memoria del microcomputer. Voi potete leggere dalla registrazione, ma non potete scrivervi altro; perciò è un componente "read only" (che legge solamente) della memoria del microcomputer.

La floppy ROM fu introdotta commercialmente per la prima volta dalla Interface Age Magazine, che la usa come mezzo per fornire ai suoi lettori per mezzo di una rivista, dei programmi direttamente sfruttabili dai computer.

Precedentemente a questo, i programmi dovevano essere stampati in linguaggio da programmazione, la qual cosa lasciava al lettore il compito di inserire il programma nel proprio microcomputer, operazione laboriosa e tale da introdurre facilmente errori.

REGISTRAZIONI CON CASSETTA

L'informazione viene immagazzinata nel nastro di una cassetta coma una sequenza di "record" fisici. Non esiste per le cassette uno schema standard di FORMAT, come invece i settori e le piste che abbiamo descritto per i floppy disk.

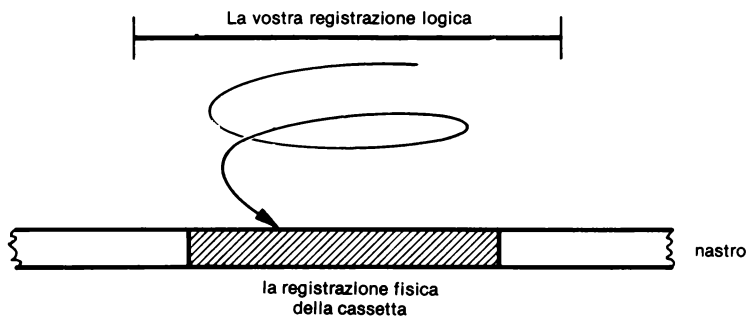
Le registrazioni fisiche su nastro da cassetta possono avere tutte la stessa lunghezza o lunghezze diverse; inoltre, non esistono restrizioni poste sulla loro lunghezza.

Un record fisico può essere lungo un solo carattere, come può essere lungo quanto la durata del nastro gli permette.

Il record logico che descrivemmo parlando di settori e piste del floppy disk può venir registrato su nastro da cassetta come un singolo record fisico.

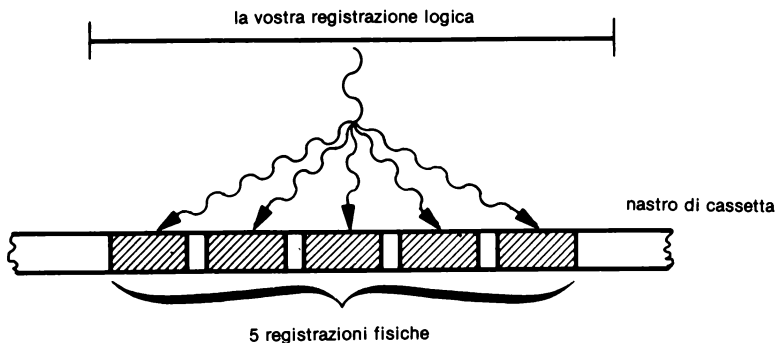
La vostra registrazione logica

nastro

la registrazione fisica
della cassetta

Oppure può essere distribuito in un certo numero di registrazioni fisiche.

la vostra registrazione logica

nastro di cassetta

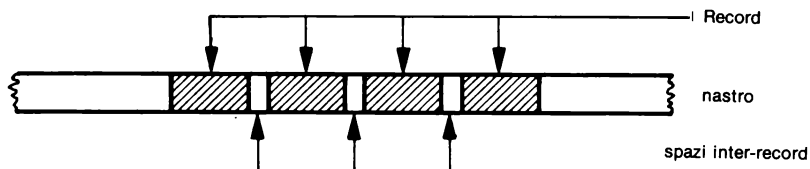
5 registrazioni fisiche

La differenza più importante fra un'unità a cassetta ed un floppy disk è che l'unità a cassetta è un dispositivo di accesso sequenziale (o seriale), mentre il floppy è un dispositivo ad accesso casuale.

Quando diciamo che una cassetta di nastro è un dispositivo sequenziale, intendiamo dire che non è possibile saltare qua e la sulla superficie del nastro allo stesso modo che è invece possibile sul floppy.

Se voi desiderate accedere al 25° record sul nastro della cassetta, dovete pazientemente attendere che siano stati superati gli altri 24 record. Quindi, informazioni stivate molto avanti nel nastro possono richiedere molto tempo per essere raggiunte, per esempio anche 10 ÷ 15 minuti.

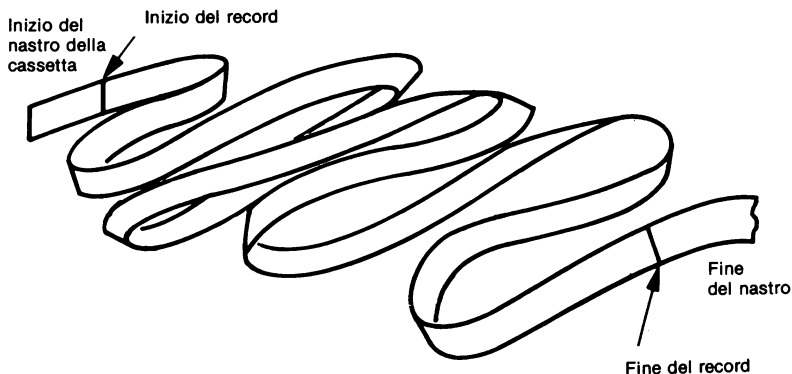
SALTI INTER-RECORD

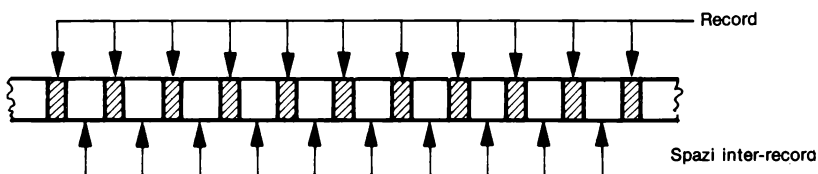
Se sul nastro della cassetta esistono più unità di registrazione, fra ognuna di esse vi saranno delle zone di separazione.

Record
nastro
spazi inter-record

Naturalmente, nei salti di separazione non esiste informazione utile.

Aumentando il numero dei record sul nastro della cassetta, aumentiamo il numero di zone vuote, e diminuiamo di conseguenza la quantità totale di informazione utile che possiano stivare sul nastro.

Quindi, per ottenere il massimo di informazione su una cassetta, dovremmo immagazzinare l'informazione come registrazione continua, senza salti inter-record.

Inizio del nastro della cassetta
Inizio del record
Fine del nastro
Fine del record

Con considerazione opposta, se voi disponete di numerosi record molto brevi, rischiate di sciupare la maggior parte delle cassette in zone vuote inter-record.

Record
Spazi inter-record

AFFIDABILITA'

Ma allora, perchè la gente dovrebbe impelagarsi con lunghe serie di brevi registrazioni? La risposta c'è, ed è: per combattere gli errori. Le cassette non sono del tutto affidabili; è facile graffiare, o comunque danneggiare, la superficie del nastro, e lo strato magnetico sul nastro tende a consumarsi con l'uso intensivo.

Per evitare errori, i sistemi microcalcolatori ben progettati registrano ogni brano di informazione due volte sul nastro della cassetta; questo è noto come registrazione ridondante.

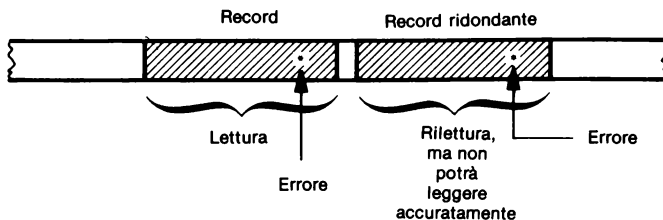
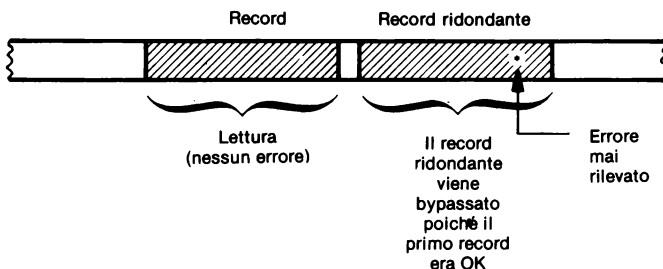
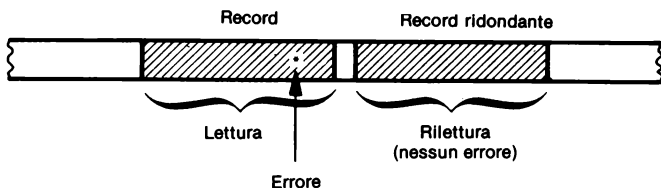
Record
Record ridondante

REGISTRAZIONE RIDONDANTE

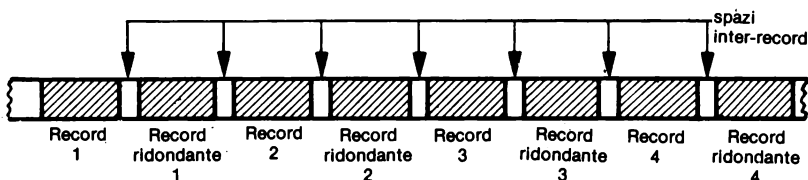
Ora supponete di avere solo una lunga registrazione per nastro; questa dovrà diventare due record se voi adottate il sistema di registrazione ridondante. In tal modo, potete anche avere uno o più errore in uno dei due record, e leggere ancora correttamente la registrazione.

Record
Record ridondante
Lettura
Rilettura
(nessun errore)
Errore

Record
Record ridondante
Lettura
(nessun errore)
Il record ridondante viene bypassato poiché il primo record era OK
Errore mai rilevato

Record
Record ridondante
Lettura
Errore
Rilettura, ma non potrà leggere accuratamente
Errore

Supponiamo ora di spezzare la registrazione unica e lunga in quattro registrazioni più corte; inoltre dotiamo ognuno di questi quattro record del suo record ridondante, ciò che porta ad avere 8 record sul nastro.

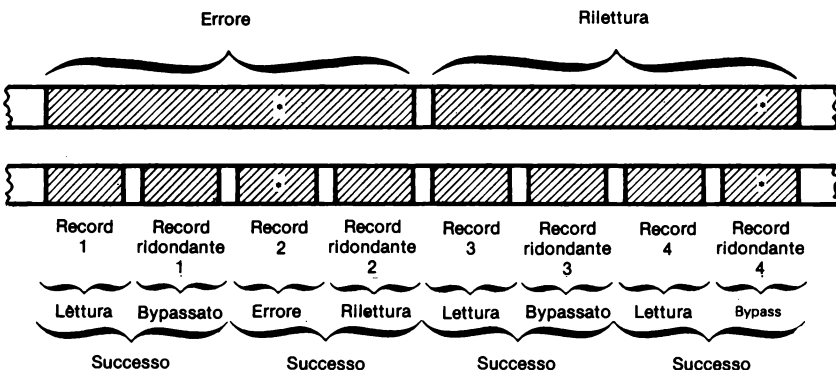
spàzi inter-record

Record 1 | Record ridondante 1 | Record 2 | Record ridondante 2 | Record 3 | Record ridondante 3 | Record 4 | Record ridondante 4

I due errori che altrimenti renderebbero inusabile la cassetta possono ora essere tollerati. Ricordando che gli errori sul nastro si verificano in quanto il nastro stesso è danneggiato sulla sua superficie, rifacendoci a quanto sopra, i due errori si verificheranno esattamente nella stessa posizione fisica del nastro.

Errore | Rilettura

Record 1 | Record ridondante 1 | Record 2 | Record ridondante 2 | Record 3 | Record ridondante 3 | Record 4 | Record ridondante 4

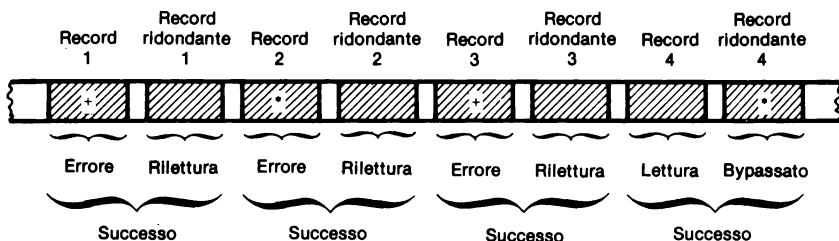
Lèttura | Bypassato | Errore | Rilettura | Lettura | Bypassato | Lettura | Bypass

Successo | Successo | Successo | Successo

La nostra cassetta è ora in grado di meglio tollerare gli errori, semplicemente in quanto abbiamo aumentato il numero dei record.

Ora possiamo tollerare due errori addizionali prima di dover buttar via la cassetta.

Record 1 | Record ridondante 1 | Record 2 | Record ridondante 2 | Record 3 | Record ridondante 3 | Record 4 | Record ridondante 4

Errore | Rilettura | Errore | Rilettura | Errore | Rilettura | Lettura | Bypassato

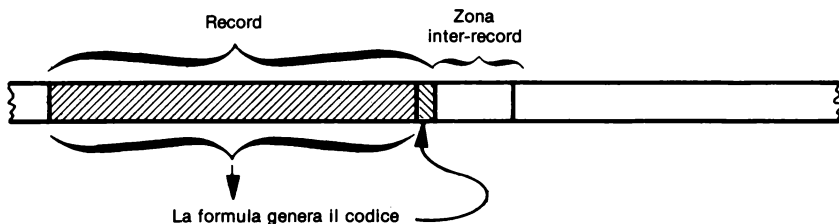
Successo | Successo | Successo | Successo

In conclusione, aumentando il numero dei record su una cassetta, diminuisce la quantità totale d'informazione che può essere registrata, ma la tollerabilità degli errori migliora – naturalmente, a patto che si usi la registrazione ridondante.

CODICE PER RIVELARE ERRORI

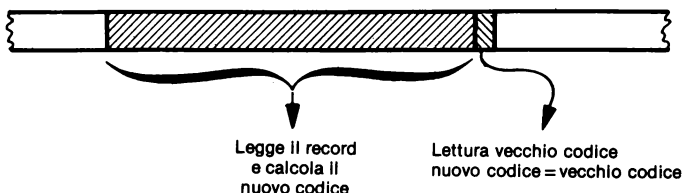
Un sistema microcomputer dà assicurazione di aver letto correttamente una registrazione scrivendo uno speciale codice di rivelazione errori alla fine di ogni record. Questo codice speciale viene calcolato applicando una formula a tutti i numeri nel record.

Record

Zona inter-record

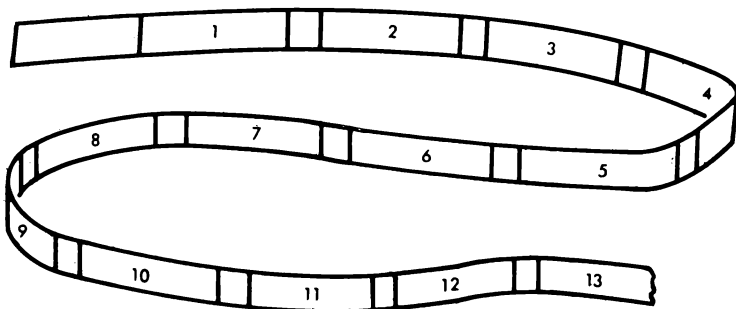
La formula genera il codice

Quando il sistema microcomputer rilegge un record, esso calcola un nuovo codice di rivelazione di errore, questa volta basato sui numeri che legge. Poi rilegge il vecchio codice di rivelazione d'errore; se il record è stato letto correttamente, il nuovo ed il vecchio codice di rivelazione di errore saranno identici.

Legge il record e calcola il nuovo codice

Lettura vecchio codice nuovo codice = vecchio codice

Se qualcuno dei numeri è stato letto in modo sbagliato, allora il nuovo codice non sarà più lo stesso del vecchio; l'informazione allora sarà giudicata sbagliata.

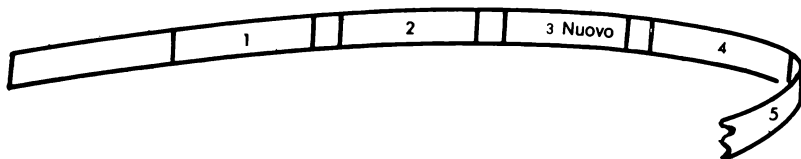
SCRITTURA E LETTURA DELLE CASSETTE

Il fatto che l'informazione su cassette debba essere raggiunta sequenzialmente rende molto difficile leggere e scrivere sulla medesima cassetta. Supponiamo, per esempio, che stiate immagazzinando una lista di nomi e indirizzi su un nastro.

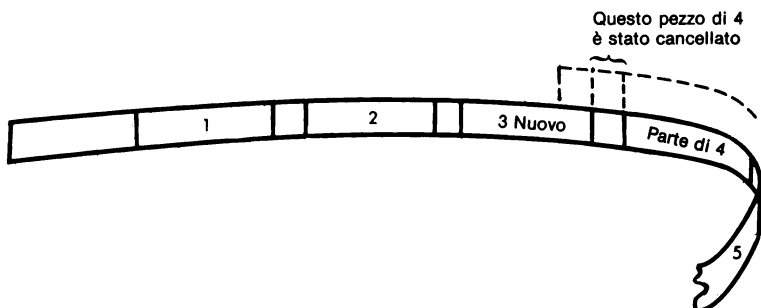
1 2 3 4 5 6 7 8 9 10 11 12 13

I numeri dall'1 al 13 rappresentano 13 indirizzi individuali

Supponiamo ora che dobbiate cambiare il terzo nome e indirizzo (3).

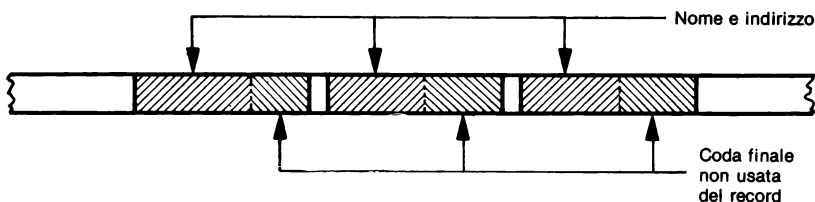
Ciò potrebbe sembrare un gioco abbastanza semplice: si tratta semplicemente di scrivere il nuovo nome e indirizzo sopra il vecchio.

1 2 3 Nuovo 4 5

Ma aspettate un momento, poichè questa potrebbe diventare un'operazione molto laboriosa. Supponiamo che il nuovo nome e indirizzo siano più lunghi del vecchio; dovrete allora sfruttare anche un pezzo del successivo nome e indirizzo.

Questo pezzo di 4 è stato cancellato

1 2 3 Nuovo Parte di 4 5

Possiamo ovviare al problema riservando lo stesso spazio per ogni nome e indirizzo senza tener conto del numero di caratteri attualmente esistenti nel nome e indirizzo. I caratteri eccedenti verranno lasciati vuoti.

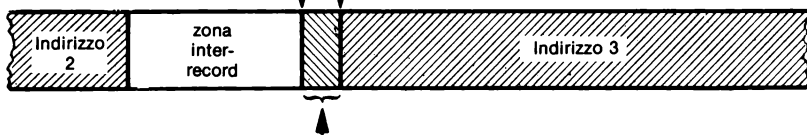
Nome e indirizzo

Coda finale non usata del record

Anche questa non è una soluzione adeguata, in quanto richiede che il meccanismo di trazione della vostra cassetta sia molto preciso. Supponiamo che partiate a riscrivere con una piccolissima distanza di ritardo.

Il vecchio indirizzo 3 partiva qui

Il nuovo indirizzo 3 parte qui

Indirizzo 2

zona inter-record

Indirizzo 3

Questo apparirà come un brevissimo record, o come parte dell'indirizzo 3

Quando il microcalcolatore legge "Indirizzo 3", egli andrà a leggere tutto il rimanente di quel nome e indirizzo, e commetterà quindi un errore di lettura.

Viceversa, se partisse troppo presto, ci sarebbe un buco alla fine del record, e di nuovo si verificherebbe un errore in lettura.

Chiaramente, lo scrivere cancellando registrazioni precedenti vuol proprio dire andarseli a cercare, i guai.

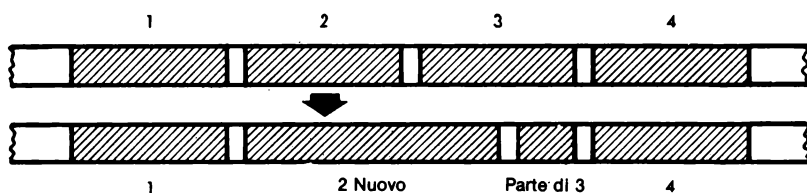
I registratori a cassetta economici daranno più guai di quelli costosi, perchè i primi hanno meccanismi di trascinamento meno precisi.

Ma il vero problema è che basta un solo errore per rovinare tutta la cassetta; e non ha importanza se questo errore si verifica frequentemente o occasionalmente: in ogni caso esso renderà la cassetta inutilizzabile.

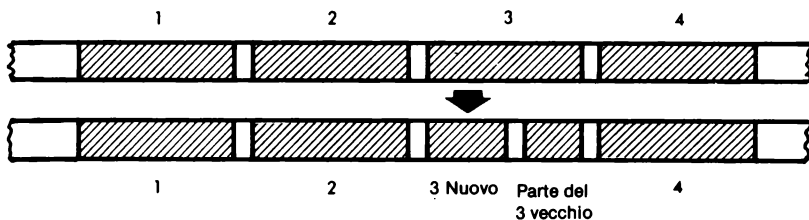
Se avete record lunghi e record corti tutti mescolati in una cassetta, allora il leggere e lo scrivere sulla medesima cassetta diventa veramente un compito senza speranza.

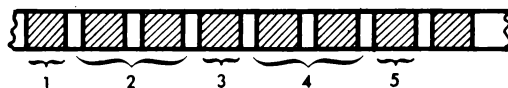
Se ogni registrazione logica è immagazzinata sul nastro della cassetta come una singola registrazione fisica, allora chiaramente non sarà possibile scrivere record di lunghezze diverse nello stesso spazio. Infatti un record più lungo andrà a interferire con il record successivo.

1 2 3 4

1 2 Nuovo Parte di 3 4

Invece una registrazione più corta lascerà parte della vecchia registrazione ... in attesa di essere rivelata come un errore.

1 2 3 4

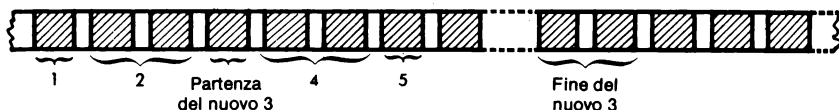
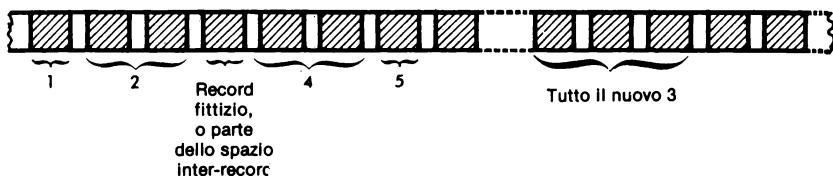
1 2 3 Nuovo Parte del 3 vecchio 4

Se registrazioni logiche di lunghezze diverse sono stivate nel nastro come sequenza di registrazioni fisiche di uguale lunghezza, si potrà scrivere e leggere sulla medesima cassetta spezzando il record logico così come abbiamo visto per il floppy disk.

1 2 3 4 5

Cominciando con questo nastro, vi sono qui due modi alternativi di inserire un nuovo record 3.

Alternativa A

1 2 Partenza del nuovo 3 4 5 Fine del nuovo 3

Alternativa B

1 2 Record fittizio, o parte dello spazio inter-recorc 4 5 Tutto il nuovo 3

Il problema con questi schemi di utilizzo è che potrete spendere la vostra vita ad avvolgere e riavvolgere cassette.

Nel caso dell'alternativa A, elenchiamo i passi che servirebbero per leggere la registrazione sopra illustrata.

1) Trovare il record logico 2;
2) Trovare l'inizio del record logico 3;
3) Avvolgere (in avanti) la cassetta per trovare la fine del record logico 4;
4) Riavvolgere fino all'inizio del record logico 4 ecc.

Usando invece l'alternativa B, finirete per occupare un sacco di spazio sul nastro. E inoltre, se i vostri record erano in un qualche tipo di ordine (per esempio,alfabetico,) essi passeranno presto in un totale disordine.

Dobbiamo concludere che l'avere solo una unità a cassette nel vostro sistema microcalcolatore può essere molto pericoloso; dovrete avere almeno due unità a cassette, una per leggere ed una per scrivere.

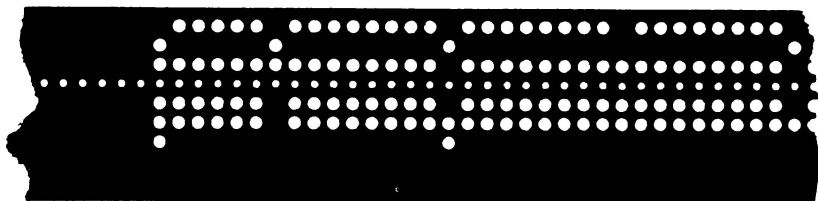
UNITA' A NASTRO DI CARTA

Esiste anche un dispositivo di memoria grossolana che è perfino più primitivo delle cassette: si tratta del nastro di carta. C'era un tempo nel quale il nastro di carta rappresentava l'unico mezzo, a basso costo, per immagazzinare l'informazione di un computer, ma oggi le cassette sono quasi altrettanto economiche e ben più veloci.

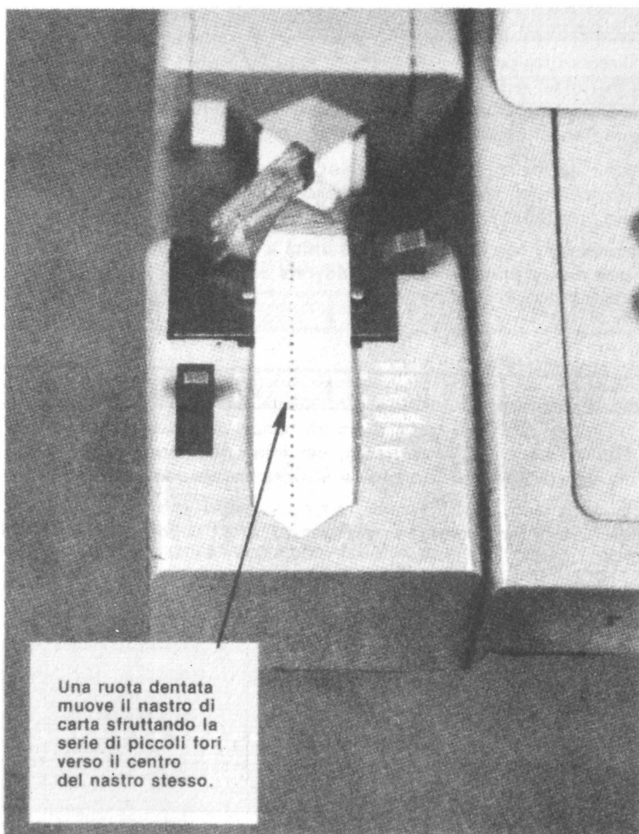
Esistono, cionondimeno, molti sistemi microcalcolatori che usano nastro di carta per immagazzinare informazioni; perciò questo è un argomento che dobbiamo discutere.

Il nastro è costituito, come dice la parola, da una sottile striscia di carta.

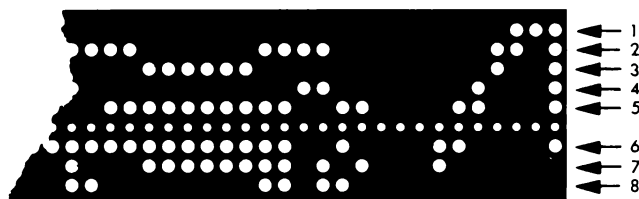
CARATTERISTICHE DEL NASTRO

L'informazione è immessa nel nastro di carta mediante perforazione di tanti piccoli fori sullo stesso. Ogni serie di fori verticali rappresenta un carattere.

FORATURA DI TRASCINAMENTO

In aggiunta ai fori che rappresentano i caratteri, **la maggioranza dei nastri in carta possiedono una linea continua di fori di trascinamento** (il cui scopo è appunto quello di far muovere il nastro di carta), punzonati pressapoco al centro del nastro.

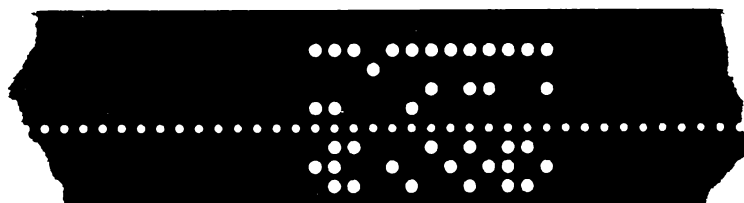
Una ruota dentata muove il nastro di carta sfruttando la serie di piccoli fori verso il centro del nastro stesso.

CANALI DEL NASTRO DI CARTA

Esistono otto posizioni su ciascuna linea verticale nelle quali può essere o non essere punzonato un foro.

1
2
3
4
5
6
7
8

Le otto posizioni dei fori si chiamano **canali**. La combinazione delle posizioni forate o non forate costituisce un codice che, in qualche modo prefissato, identifica un carattere.

Quando parliamo di carattere, ci riferiamo qui a ciascuna lettera dell'alfabeto, a ciascun digit numerico, o ad ogni carattere particolare come la virgola, il punto, gli esclamativi ed interrogativi, ecc.

Vi sono dieci serie di fori per pollice di nastro, quindi un pollice può registrare dieci caratteri di informazione.

Per esempio, il nome JOE BITBURGER richiede 35 mm di nastro, se registrato come ora detto.

Lo svantaggio principale del nastro di carta è che ne servono quantità enormi.

Nonostante che le densità di immagazzinamento delle cassette varino molto, sare costretti ad usare circa 600 m di nastro di carta per registrare la stessa quantità di informazione che sta in una sola cassetta da 90 minuti o su una sola facciata di un floppy disk a singola densità.

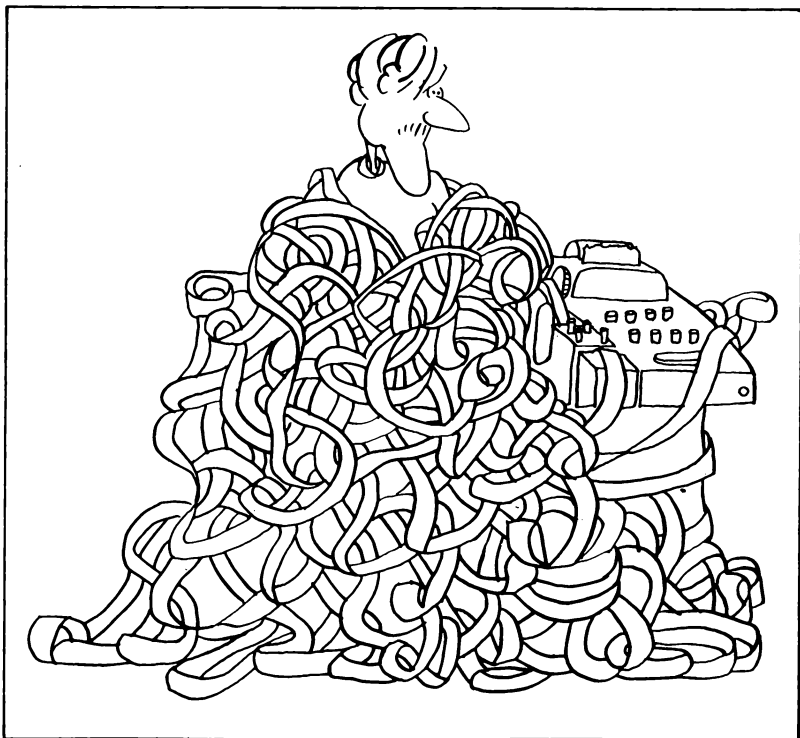
Un altro svantaggio del nastro di carta è che esso richiede un tempo piuttosto lungo (relativamente parlando) per registrare o leggere l'informazione.

I tempi di lettura e scrittura variano tipicamente fra 10 e 100 caratteri al secondo.

Nonostante ciò possa sembrare piuttosto veloce, le cassette possono essere scritte e lette fra 100 e 1000 caratteri al secondo, mentre le velocità tipiche dei floppy disk variano fra i 1000 ed i 10.000 caratteri. Se poi ci si riferisce ai dischi rigidi, essi possono lavorare ad oltre 1 milione di caratteri/secondo.

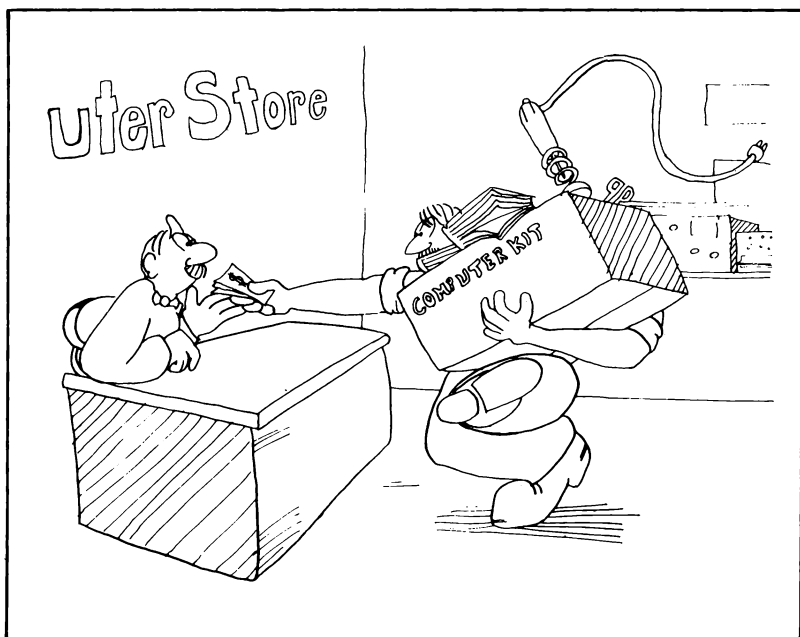
Parlando del tempo, bisogna considerare il metro totalmente diverso con cui esso viene considerato quando si comincia a lavorare coi computer.

Perchè, infatti, è considerato lento un ritmo di 10 caratteri al secondo? La ragione sta nel fatto che usualmente si leggono o scrivono centinaia o migliaia di caratteri alla volta; e più spesso di quello che pensiate, capita che non abbiate altro da fare che guardare mentre si sta effettuando l'operazione di lettura o scrittura.

E allora anche 10 secondi diventano un tempo lungo da attendere se vi trovate a dover far questo frequentemente.

Capitolo 2

USATE UN MICROCOMPUTER E GUARDATELO CRESCERE

Esaminiamo ora i molti modi nei quali voi potete usare un sistema microcomputer. Per farci aiutare in questa impresa, andiamo a cercare Joe Bitburger, un intrepido hobbista di microcomputer.

Avendo lavorato occasionalmente con i computer, un bel giorno il nostro Joe è colpito da un attacco di irrazionalità, durante il quale egli acquista un microcomputer al locale magazzino.

Il comportamento di Joe non è inusuale, e non è neanche molto ragionevole; il nostro Joe ha ancora molto da imparare.

Dicendo che Joe lavora occasionalmente coi computer, si intende che, nel corso del suo lavoro, egli occasionalmente scrive programmi per computer usando un linguaggio da computer chiamato FORTRAN.

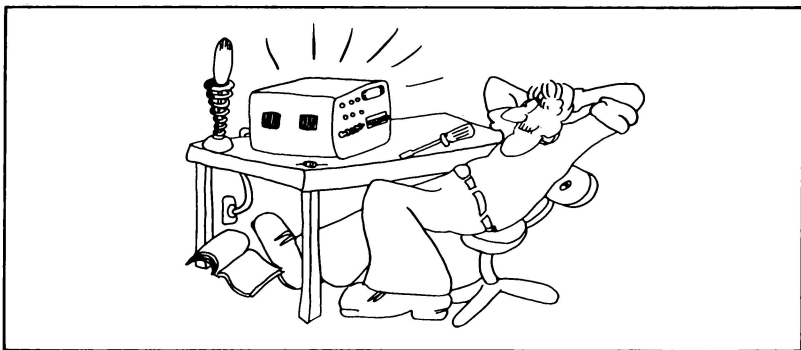
Ogni mattina Joe coscienziosamente porta una pila di carte (carte da computer, non carte da gioco!) al centro elaborazione del suo ufficio.

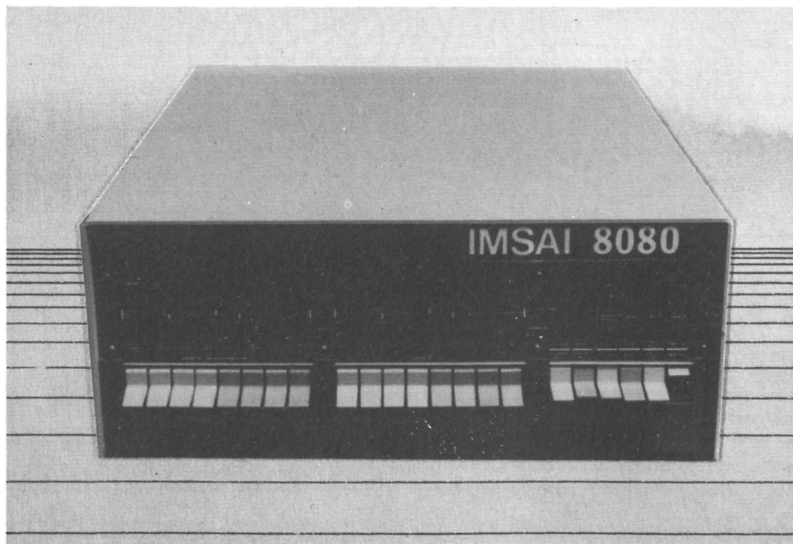
Se tutto va bene, talvolta subito dopo pranzo egli si ferma di nuovo al centro computer per prendere un grosso foglio di carta ripiegata sulla quale sono stampati i suoi risultati.

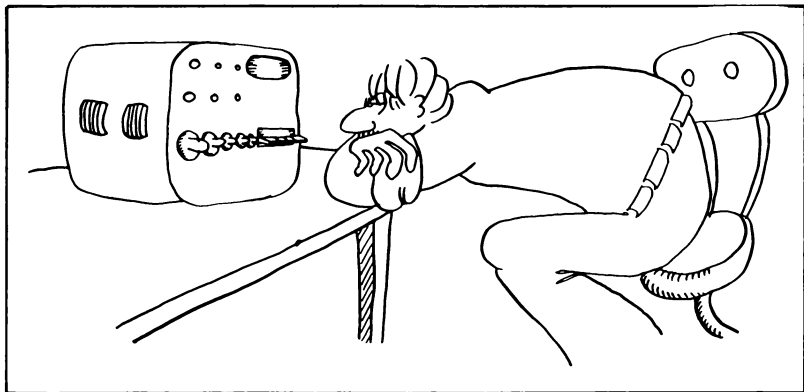
Forse, una delle cose che hanno guidato Joe nel suo atto irrazionale è stato il fatto che, pur avendo scritto programmi per computer per oltre due anni, egli non è mai stato abbastanza vicino ad un computer da riuscire a toccarlo.

Joe porta orgogliosamente a casa il suo microcomputer — per montarlo e usarlo —. A questo punto, noi dobbiamo saltare molte settimane (o addirittura mesi) della vita di Joe Bitburger.

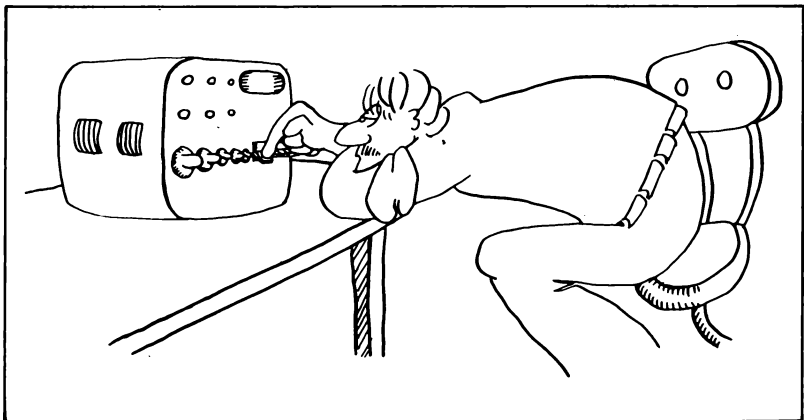
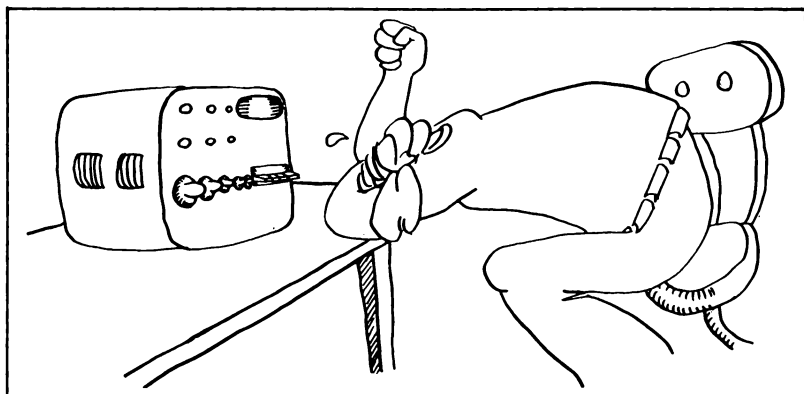
Sappiamo che Joe aveva portato a casa il suo Kit di microcomputer. Come molti hobbisti di computer potranno riferirvi, il processo di assemblaggio di tale Kit vi causerà molte ore di ansia, durante le quali metterete assieme molti dubbi sui parenti di chi ha realizzato il Kit e di chi ne ha scritto il manuale.

Finalmente, dopo molti salti al magazzino e molti rimbrotti da parte del proprietario per saldature malfatte, istruzioni malintese e cose simili, **Joe Bitburger riesce a far funzionare il suo microcomputer.**

Povero Joe: solo la soddisfazione di aver portato a termine una grossa avventura lo rende felice. **Tutto quello che ha è una scatola con dentro un microcomputer.**

IMSAI 8080

CREARE UN PROGRAMMA E FARLO OPERARE

Avendo costruito il suo microcomputer, Joe desidera farci qualcosa. Ma quello che può fare è favolosamente poco. Avendo poche scelte, **Joe decide di scrivere un programma che faccia accendere una lampadina sul pannello frontale.**

Joe scrive il suo programma su un pezzo di carta usando un "linguaggio di programmazione", che ha dovuto imparare per l'occasione. Quindi egli converte il programma in una sequenza di numeri. Ricordate, tutti i programmi diventano, prima o poi, una sequenza di numeri, in quanto questa è l'unica maniera in cui il microcomputer può capire un programma. Avendo creato tale sequenza di numeri, Joe li deve caricare nella memoria del microcomputer. Ma Joe non ha una tastiera, non ha nient'altro se non il microcomputer. Cosa farà allora in tale circostanza?

IL PANNELLO FRONTALE DI UN MICROCOMPUTER

Il microcomputer che Joe ha comprato ha un pannello frontale con interruttori e spie. **Joe impara come far entrare i numeri nella memoria del microcomputer premendo gli interruttori sul pannello frontale nella sequenza corretta.**

Dopo 20 minuti, ed i calli alle dita, Joe ha finito; con impazienza **egli mette all'opera il microcomputer. Ma, chissà perchè non funziona.**

E' così che Joe decide di esaminare l'informazione realmente immagazzinata nella memoria del microcomputer. Per far questo, Joe usa ancora gli interruttori e le spie sul pannello frontale.

Si tratta degli stessi interruttori e delle stesse luci che Joe ha già usato per caricare l'informazione nella memoria del microcomputer. Concettualmente, questo può essere illustrato come segue.

Ora l'informazione che Joe ha inserito torna fuori come luce di queste spie e Joe usa gli interruttori per selezionare la locazione della memoria

Joe ha usato questi interruttori per inserire le informazioni nella memoria del microcomputer

Questi interruttori permettono a Joe di caricare l'informazione nel microcomputer, rileggere l'informazione o effettuare altre operazioni

FUNZIONI DEL PANNELLO FRONTALE

Non tentate di capire esattamente come operano gli interruttori e le spie sul pannello frontale; potreste inutilmente confondervi le idee, e nient'altro. Il pannello frontale illustrato qui sopra è diverso da quelli che appaiono in

ciascuna fotografia; sotto tale aspetto, si può dire che non esistano due pannelli frontali uguali, e addirittura alcuni microcomputer neanche ne hanno uno.
I pannelli frontali vengono usati per caricare le informazioni nella memoria del microcomputer, per esaminare i contenuti della memoria del microcomputer e per controllare le operazioni del microcomputer in generale.

Allo scopo di esaminare il contenuto della memoria del microcomputer, Joe legge semplicemente il suo manuale e segue le istruzioni passo passo. Ciò è quanto anche voi fareste, se vi trovaste nelle stesse condizioni di Joe.

Cosa ha trovato Joe? Due interruttori che dovevano essere aperti, sono chiusi, ed uno che doveva essere chiuso è aperto.

Per niente intimidito, Joe sistema gli interruttori nella posizione giusta e riprova. Ma ancora una volta, il programma non funziona.

Così, ancora una volta Joe passa attraverso il laborioso processo di controllare cosa c'era nella memoria del microcomputer.

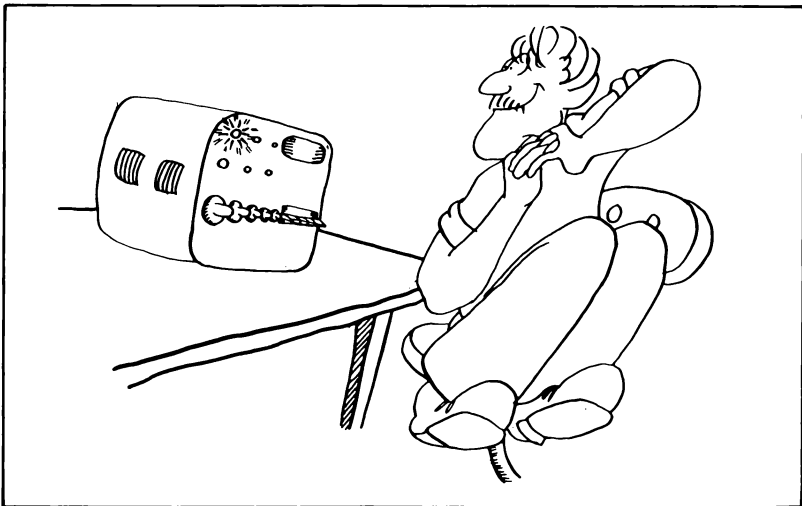
Tutto è esattamente come dovrebbe, ma il programma non funziona: perché? **Devono essere sbagliate le sequenze di numeri che rappresentano il programma.**

Così Joe torna al suo programma; controllandolo accuratamente egli scopre che vi sono realmente gli errori.

PROGRAMMA DI DEBUG

Quello che Joe ha fatto si chiama il "debugging" di un programma. Gli errori individuali sono chiamati "bug" di programma.

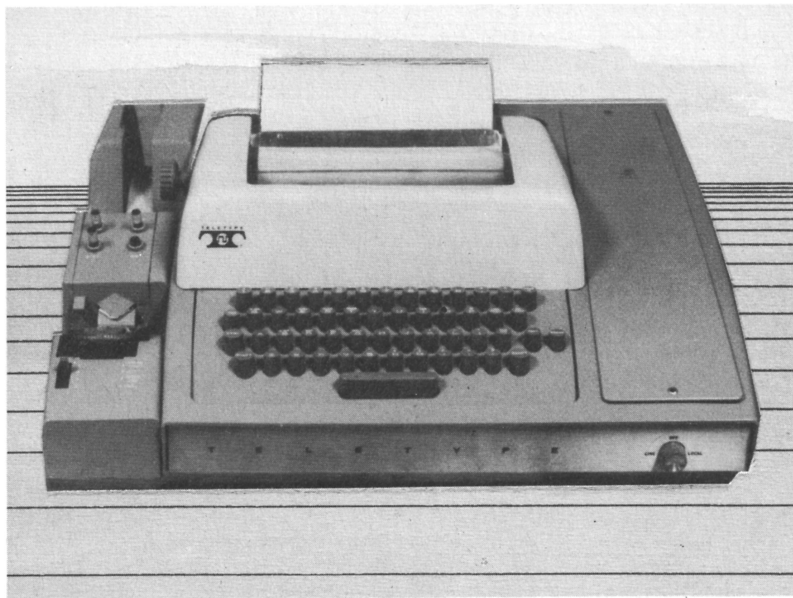
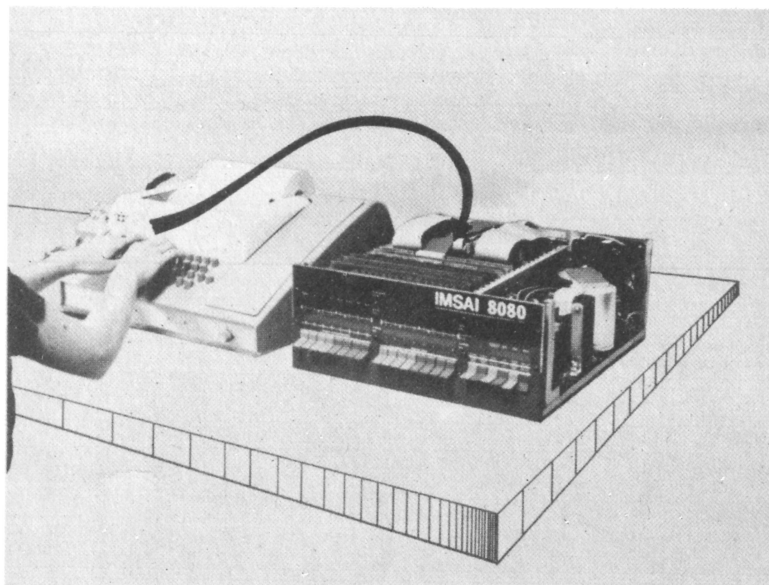
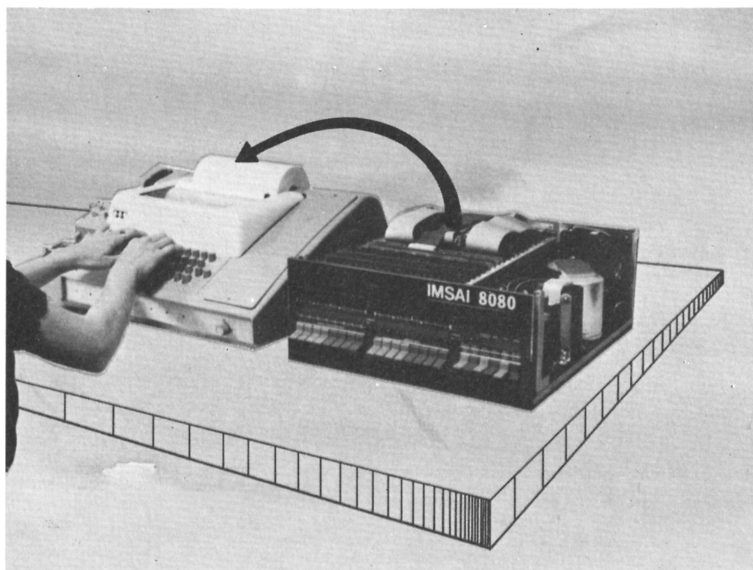
Questa seguenza consistente nel correggere gli errori di posizione degli interruttori e quindi di ricercare ulteriori errori nel programma, può verificarsi diverse volte.

Infatti, è dopo due giorni che Joe finalmente riesce a far funzionare il suo programma. **Trionfalmente, egli siede di fronte al suo microcomputer, osservando una luce che si accende sul pannello.**

Joe ha speso centinaia di dollari e di ore, solo per riuscire a vedere una luce che si accende su un display?

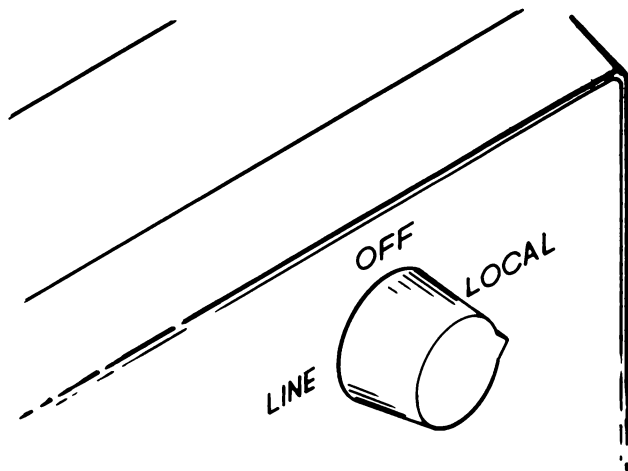
C'è veramente una grossa differenza dalle grandi cose che avvengono al suo ufficio, fra il momento in cui Joe lascia la sua pila di carte da programma al centro di calcolo ed il momento in cui egli ritira un pacco di risultati stampati.

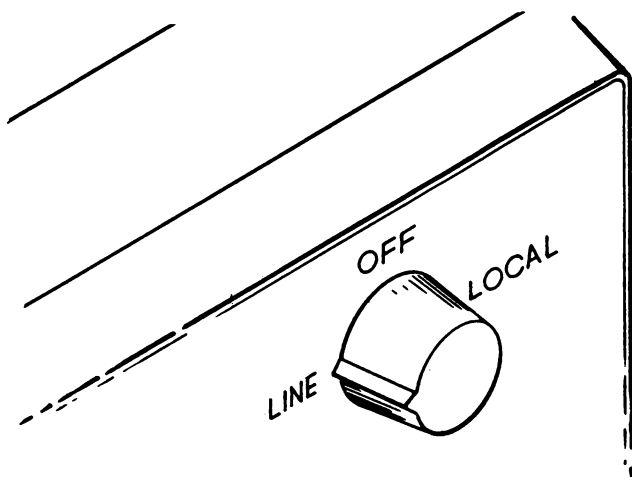
Chiaramente, Joe deve fornire il suo microcomputer di occhi ed orecchie; allora Joe prende a prestito un terminale per telescrivente da un amico.

IL TERMINALE PER TELESCRIVENTE

Il terminale da teletype è un dispositivo molto interessante; è in giro da più di 20 anni, virtualmente inalterato. Il motivo della sua popolarità, e del suo successo, sta nel fatto che esso fornisce un po' di tutto quanto serve per completare un microcomputer e raramente si guasta. Diamo un'occhiata ad un terminale per telescrivente.

Esso ha una tastiera che voi potete usare per inserire l'informazione nella memoria del vostro microcomputer.

Il terminale teletype ha anche una stampante, che può essere usata per stampare l'informazione in uscita, o per portare avanti un dialogo con il microcomputer.

La stampante del terminale da telescrivente serve per le funzioni combinate di video display e stampante; ma la stampante non è necessariamente collegata alla tastiera.

TELETYPE IN "LOCALE"

E' possibile, volendo, usare un terminale da telescrivente come una semplice macchina da scrivere, cioè scollegata dal microcomputer. C'è un commutatore in fondo alla tastiera, che, quando è in posizione "LOCAL", fa lavorare il terminale nel modo suddetto. Quando opera in "locale" il terminale per telescrivente stampa un carattere ogni volta che un tasto viene battuto.

TELETYPE IN "LINE"

Se invece è inserito in "linea", il terminale è sotto il controllo del microcomputer.

L'ECO

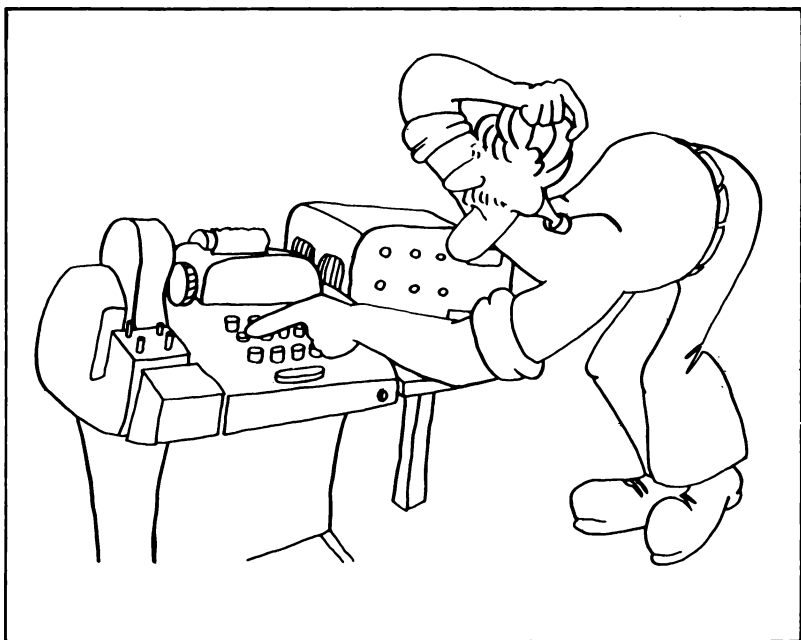
Quando spingete un tasto, un codice rappresentativo del tasto che avete premuto verrà trasmesso al microcomputer, il quale leggerà il codice e lo immagazzinerà in memoria, facendo in modo che un programma appropriato venga eseguito non appena voi premete il tasto.

Se il programma che riceve l'input del tasto fa stampare il carattere che voi avete premuto, allora il terminale teletype risponde sostanzialmente come farebbe una macchina per scrivere. Ricorderete che abbiamo parlato di ciò come di effetto "eco"'

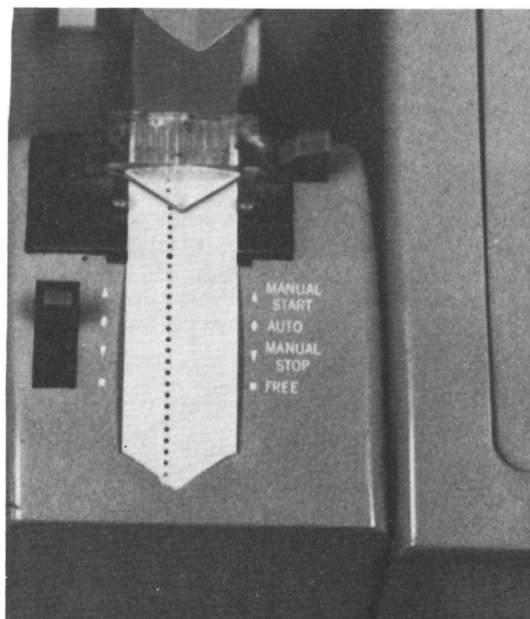
Ma se il programma che riceve l'informazione che voi introducete con la tastiera non rimanda indietro questa informazione alla stampante, allora non si verifica la ristampa, cioè non si ha l'eco.

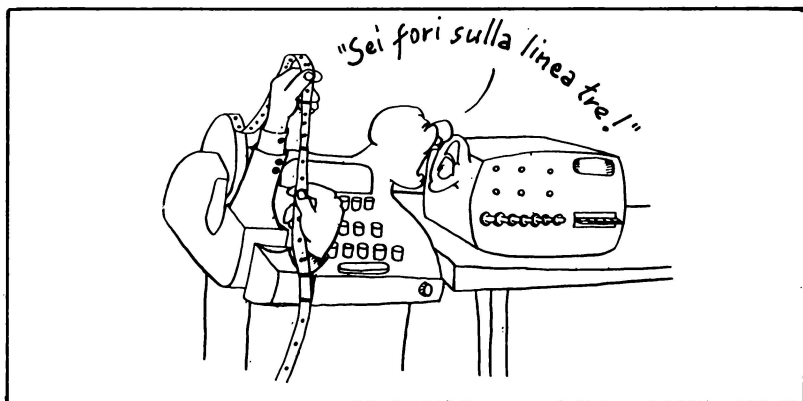
Se poi desiderate essere spiritosi, potreste scrivere un programma che riecheggi un carattere diverso da quello che voi azionate.

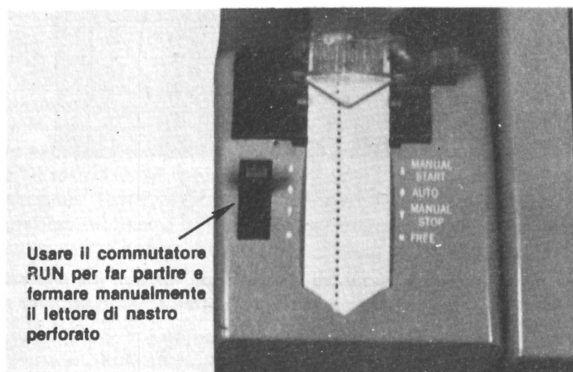
Per esempio, voi potreste far stampare come eco la lettera successiva nell'ordine alfabetico per esempio, B se si è battuto A, C se si è battuto B, e così via.

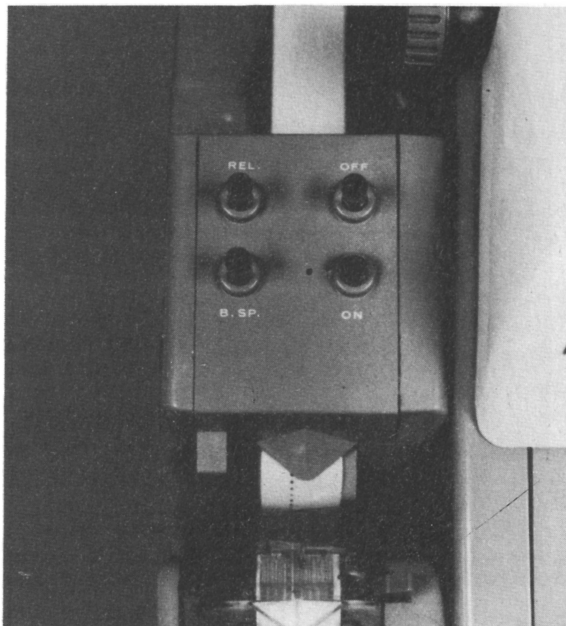
Tutto dipende da cosa avete programmato che il vostro microcomputer faccia. Se premete un tasto della telescrivente mentre il vostro microcomputer sta eseguendo un programma che non aspetta segnale d'ingresso dal terminale stesso, allora il dato che voi inserite cade in orecchie sorde.

Se il microcomputer non sta eseguendo un programma che aspetti dati dal terminale Teletype, allora voi potete premere tutti i tasti che volete, ma non succederà nulla. L'informazione verrà completamente ignorata dal microcomputer e nulla verrà stampato.

LETTORE DI NASTRO PERFORATO

Alcuni terminali di telescrivente sono dotati di lettore di nastro perforato.

Come il nastro di carta si muove attraverso il lettore, questi rivela la presenza o l'assenza di fori su ciascuna linea verticale e manda un appropriato codice al microcomputer.

"Sei fori sulla linea tre!"

Esiste un interruttore che potete usare appunto allo scopo di far partire il lettore di nastro perforato.

Usare il commutatore RUN per far partire e fermare manualmente il lettore di nastro perforato

Alcuni programmi che usano il lettore di carta perforata aspettano semplicemente che voi diate la partenza al lettore, inserendone l'interruttore.

Programmi più complicati possono commutare in operazione il lettore per conto vostro. **Alcuni terminali per telescrivente hanno pure il perforatore di nastri; su esso esistono, in genere, 4 pulsanti.**

STAMPANTE E PERFORATORE

Gli interruttori ON e OFF inseriscono o disinseriscono come dicono i loro nomi, il perforatore di nastro di carta. Questi interruttori sono necessari al perforatore in quanto il **terminale teletype considera la stampante ed il perforatore come fossero fisicamente la stessa unità.**

Se il perforatore è inserito, allora qualsiasi cosa stampata viene anche perforata sul nastro. Analogamente, qualsiasi cosa perforata sul nastro verrà sempre stampata.

Così stando le cose, lascerete il perforatore disinserito finché non vorrete effettivamente perforare il nastro; **allora scriverete un programma che vi dia il tempo opportuno per mettere in funzione il perforatore.**

La tipica logica del programma può essere concettualmente illustrata come segue:

Stampare il messaggio dicendo all'operatore di inserire il perforatore di nastro

↓

Far entrare un carattere dalla tastiera della telescrivente. Non provocare eco.

La logica di programma qui illustrata può provocare, da parte della stampante teletype, la stampa di un messaggio che vi dica di inserire il perforatore. Il programma quindi si ferma finché voi non premete un tasto qualunque sulla tastiera della telescrivente. In tal modo avete a disposizione tutto il tempo che volete per inserire il perforatore. Il tasto che voi premete sulla tastiera non può produrre eco, perché i caratteri "echeggiati" sarebbero stampati e perforati. La perforazione del carattere in eco creerebbe, sul nastro di carta, una serie spuria di fori, precedendo l'informazione reale che voi volete avviare all'uscita.

Una volta che voi abbiate terminato la perforazione del nastro di carta, il vostro programma deve ancora una volta darvi il tempo di disinserire il perforatore.

Questa volta, però, voi non potete far stampare un messaggio che dica all'operatore di disinserire il perforatore, in quanto qualunque messaggio di questo tipo verrebbe esso pure perforato sul nastro.

Perciò il sistema microcomputer semplicemente si fermerà; dovete sapere che si suppone siate voi a disinserire il perforatore, e poi a spingere un tasto qualunque della tastiera allo scopo di continuare l'esecuzione del programma.

Perchè ci preoccupiamo di tutte queste laboriose precauzioni, semplicemente per avere il perforatore inserito per qualche breve periodo di tempo? La risposta è: per semplice convenienza.

Ci fa comodo usare la stampante teletype per molti scopi; per stampare i risultati e per stampare il dialogo durante l'entrata dei dati e le normali operazioni del computer.

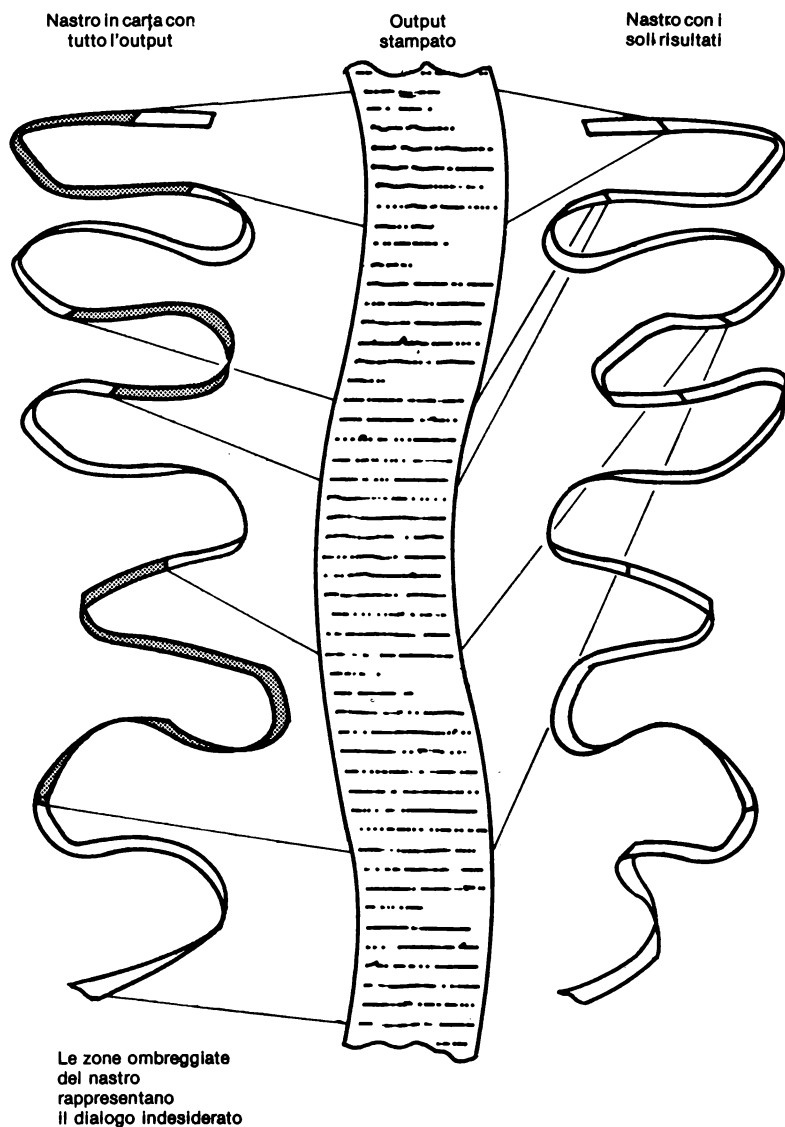
Se il perforatore è inserito permanentemente, voi dovrete di conseguenza scegliere

la vostra strada attraverso il nastro, identificandone quei pezzi che contengono risultati che vi interessa conservare, e quelle zone che invece contengono dialogo e zavorra.

Nastro in carta con tutto l'output

Output stampato

Nastro con i soli risultati

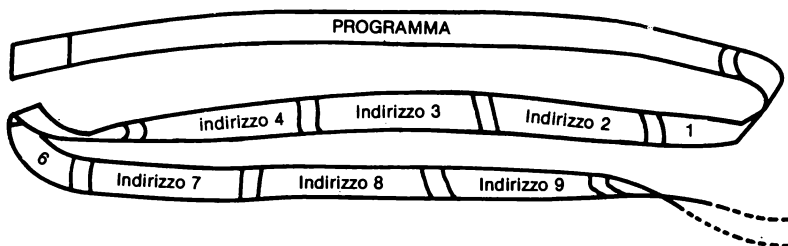
Le zone ombreggiate del nastro rappresentano il dialogo indesiderato

PULSANTE REL. DELLA TELETYPE

PULSANTE B. SP DELLA TELETYPE

Ci sono altri due pulsanti di controllo sul perforatore. Uno di essi è il pulsante di sblocco, o RELEASE (REL); esso vi permette di far slittare il nastro di carta avanti e indietro quando lo state inserendo nel perforatore.
L'altro è il pulsante di spaziature o BACKSPACE (B. SP.); esso permette di far retrocedere il nastro, una posizione di carattere per volta.

L'USO DI UN SEMPLICE SISTEMA MICROCOMPUTER

Permessosi il lusso di un terminale per telescrivente, ora Joe può far fare al suo microcomputer delle cose utili. La prima cosa utile che Joe fa è quella di scrivere un programma che lo aiuti a pagare i suoi debiti; ciò, fra l'altro, è molto pertinente, in quanto il microcomputer ha provocato una quota di debiti superiore al normale.

Ora, come potrà un microcalcolatore riuscire ad aiutare Joe a pagare i suoi conti? Semplicemente, esso può evitare a Joe il crampo dello scrittore.
Il programma di Joe contiene una lista di nomi e indirizzi per ciascune delle ditte che a rotazione gli chiedono pagamenti; la compagnia dei telefoni, la finanziaria, l'ente televisivo, tanto per fare alcuni nomi. **Joe ha eseguito il suo programma sul nastro di carta, il quale così appare concettualmente:**

Ora però, non occorre molto a Joe per comprendere che stare a mescolar programmi per controllare il terminale teletype è uno spreco di tempo.
Ricordiamo che è richiesta l'esecuzione di un programma nel microcomputer per accettare l'informazione introdotta alla tastiera o mediante il lettore di nastro in carta, e per mandare opzionalmente in eco questa informazione alla stampante.
Quando Joe collega per la prima volta la sua telescrivente al suo microcomputer, non c'è alcun programma (nel microcomputer) che si prenda cura del terminale teletype. Cosicchè la prima cosa che Joe deve fare è scrivere tale programma ed inserirlo nella memoria del computer azionando gli interruttori sul pannello frontale.

Il problema con tale procedura è che, ogniqualvolta Joe spegne il microcomputer, egli svuota anche completamente la memoria del microcomputer.

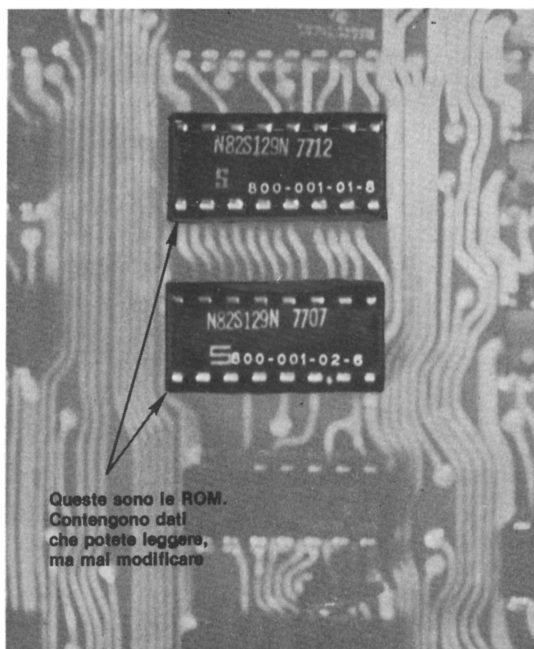
Così, tutte le volte che Joe riaccende il suo microcomputer, egli deve ripassare attraverso la laboriosa procedura di inserire il programma di controllo del terminale teletype per mezzo degli interruttori frontali.
Ancor prima che Joe abbia finito di guardare al suo programma di controllo del terminale teletype come un programma vero e proprio, esso è diventato una parte necessaria del sistema microcalcolatore. Joe torna al grande magazzino a chiedere aiuto; e lo ottiene.

MEMORIA A SOLA LETTURA

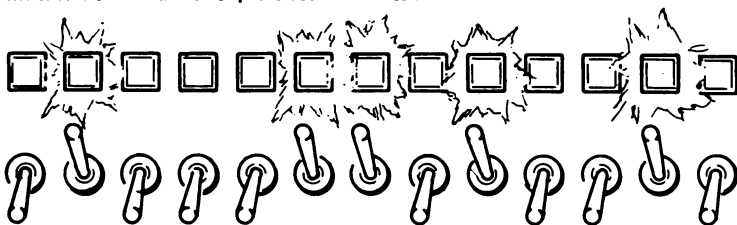
BOOTSTRAP LOADER

Chiunque abbia un terminale da telescrivente ha bisogno di un programma per controllarlo. Joe scopre che un tale programma è disponibile, caricato in una piccola piastrina di memoria che non può mai perdere il suo contenuto.

Il programma si chiama "caricatore a bootstrap", Joe ne compra uno, che ha il seguente aspetto.

Queste sono le ROM.
Contengono dati
che potete leggere,
ma mai modificare

Il termine "bootstrap" deriva dal concetto di un uomo che si solleva da solo fuori da un buco aggrappandosi ai lacci delle sue scarpe.

Mediante il programma bootstrap, il computer parte da solo. Il programma bootstrap è immagazzinato in una piastrina di memoria a sola lettura, che (come implica il suo nome) è un dispositivo di memoria che può solamente essere letto, senza potervi scrivere alcunchè. Ciò significa che il contenuto di un "chip" di memoria a sola lettura viene prefissato all'atto della costruzione e non può più essere modificato.

Allo scopo di capire la differenza fra la memoria a sola lettura e la memoria legge/scrive, immaginiamo che la piastrina di memoria sia costituita da migliaia di commutatori con una luce sopra ciascuno di essi.

Il chip di memoria è concettualmente equivalente a questi commutatori e luci spia; in realtà la microscopica struttura elettronica che copre la piastrina di memoria non ha nulla di neanche lontanamente somigliante.

Quando voi scrivete in un dispositivo di memoria, quello che state facendo può essere assimilato all'azionamento di ben precisi commutatori; una luce si accende sopra ciascun commutatore inserito.

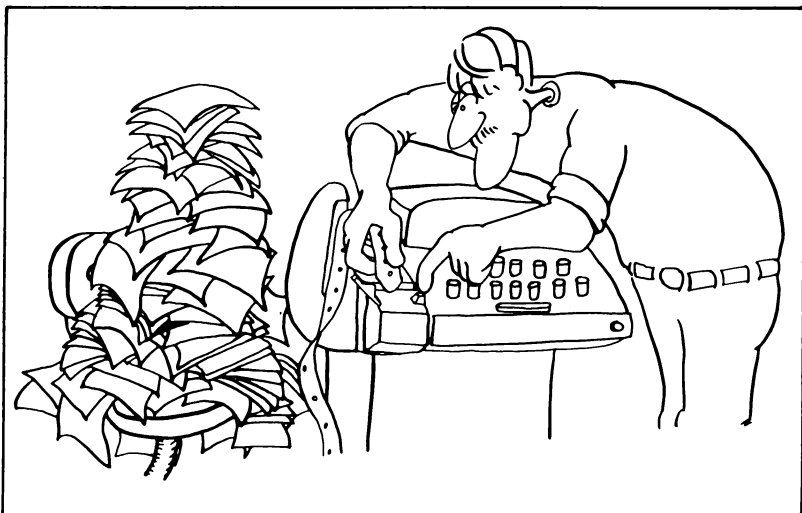
Quando voi leggete un'informazione da un dispositivo di memoria, quello che state facendo è equivalente ad esaminare quali delle luci spia sono accese e quali spente.

Supponete ora di scrivere nell'unità di memoria con l'azionare i commutatori inserendoli o disinserendoli; poi, quando siete certi che tutti i commutatori siano nella loro posizione giusta, eliminateli o rompeteli in modo che le luci che sono accese, restino accese per sempre (ed altrettanto valga naturalmente per quelle spente).

Concettualmente, è ciò che si verifica in un dispositivo di memoria a sola lettura. Il vantaggio di tali dispositivi è che essi mantengono l'informazione contenuta qualunque cosa venga loro fatta.

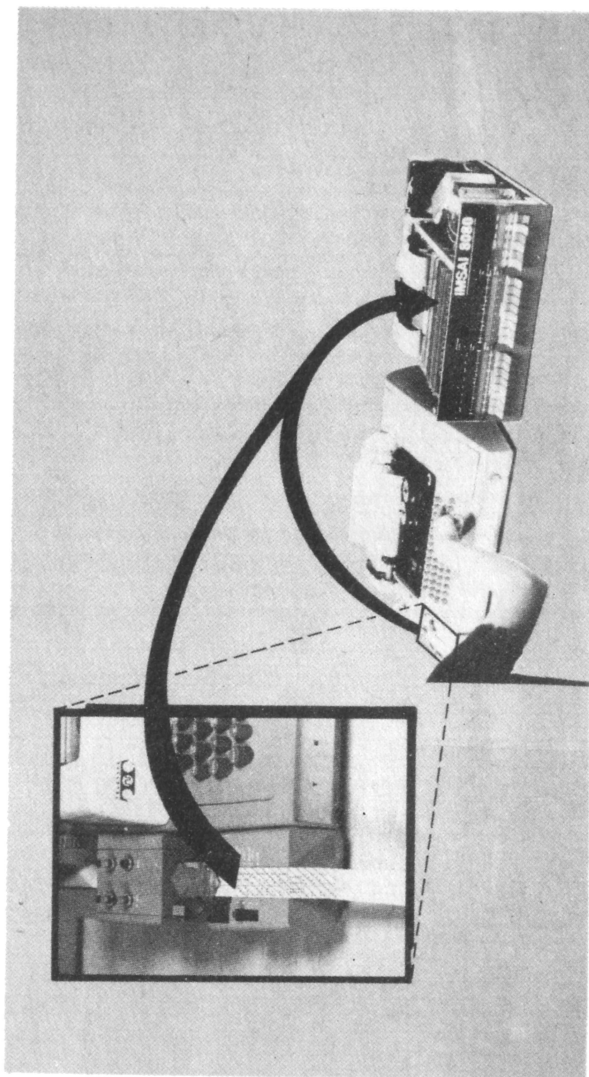
ROM

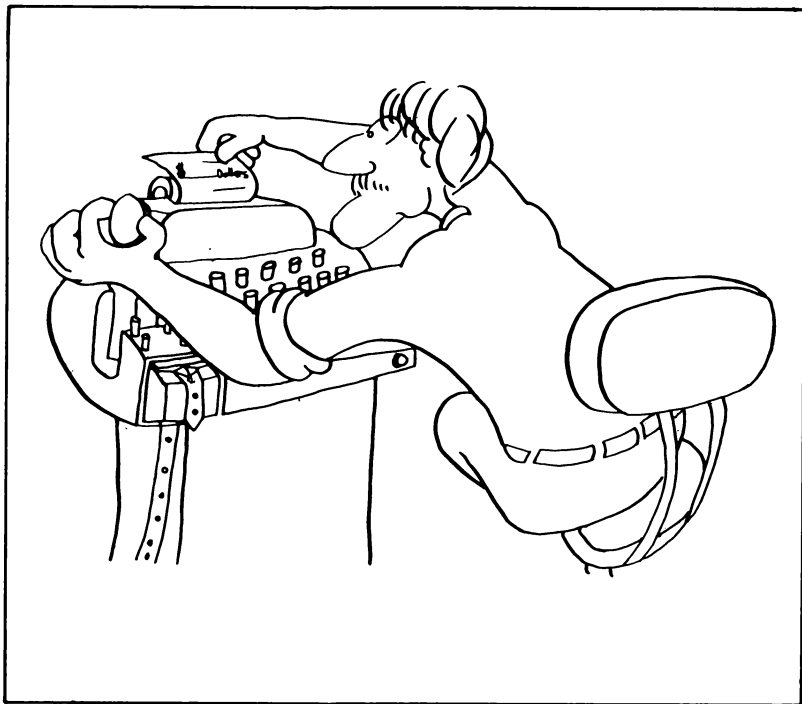
Un dispositivo di memoria a sola lettura (READ ONLY MEMORY) è normalmente indicato con le iniziali della sua definizione: ROM.

Ora, quando Joe deve pagare i suoi conti, egli fa una bella pila dei conti che deve pagare, quindi carica il nastro contenente il programma dei pagamenti da fare nel lettore di nastri (in carta) della teletype.

Subito dopo, Joe aziona alcuni interruttori, seguendo accuratamente le istruzioni ed il programma bootstrap delle ROM comincia a lavorare. Una per una, le istruzioni del programma bootstrap si gettano nel microcomputer, facendogli inserire il lettore di nastro, e quindi leggere il nastro stesso.

Il programma bootstrap non è stupido; anzi, esso è abbastanza intelligente da sapere quando ha raggiunto la fine del programma già all'inizio del nastro di carta; a quel punto egli disinserisce il lettore.

Il programma di Joe ora è stato letto nella memoria del microcomputer. Il pro-

grammatore bootstrap, sapendo che il suo lavoro è fatto, **abbandona il controllo al programma di Joe.**

Ora. le istruzioni dal programma di Joe sgusciano una dopo l'altra nel microcomputer, facendogli eseguire gli ordini di Joe.

Inizialmente, a Joe non serve che il suo programma faccia alcunché; anzi, esso deve aspettare finché Joe sta inserendo un rotolo di assegni nella stampante della teletype.

ENTRATA DALLA STAMPANTE SENZA ECO

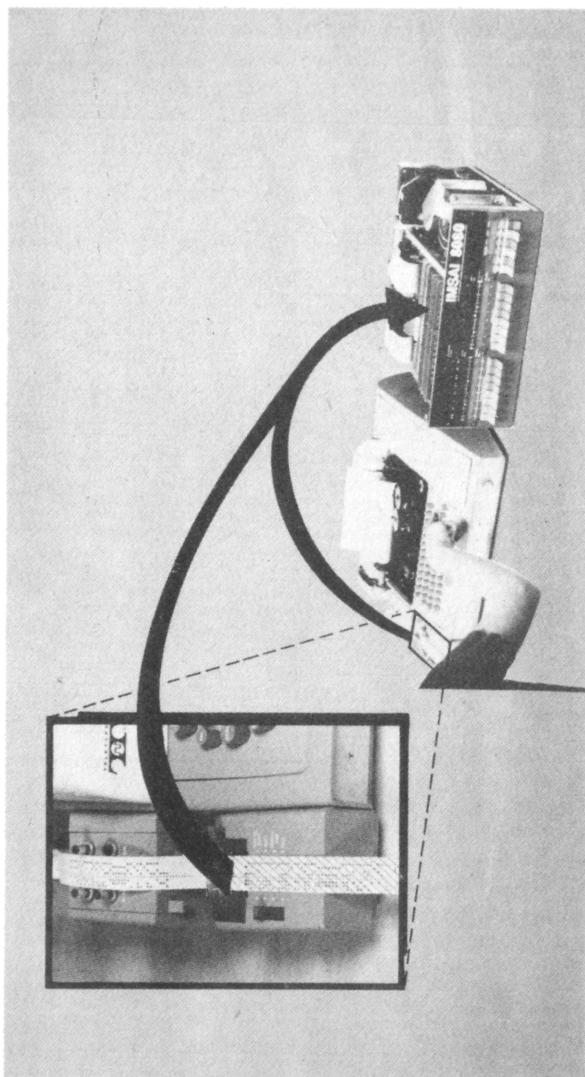
Allo scopo di dargli il tempo di battere gli assegni sulla telescrivente, il programma di Joe ha una logica molto simile a quella che inserisce o disinserisce il perforatore. **Il solo modo con cui il programma di Joe può sapere che gli assegni sono nella stampante è che Joe prema qualche pulsante della tastiera. Egli scrive che il suo programma aspetti l'input dalla tastiera della telescrivente, senza eco.**

Chiaramente, se egli preme un tasto, e il programma riecheggia il carattere, il carattere stampato apparirà su un assegno.

Joe aggiunge un completamento ai passi di programma che gli danno tempo di caricare gli assegni nella telescrivente.

Il programma di Joe è predisposto per controllare il carattere immesso tramite la tastiera, e vedere se esso è la lettera A. Solamente se è immessa la lettera A, il programma continua; per qualunque altra lettera, il programma resterà semplicemente in attesa di un'altra lettera dalla tastiera.

Ora, se Joe incidentalmente preme un tasto qualunque della telescrivente, egli non potrà far ripartire inavvertitamente il programma. Solo premendo la lettera A il programma ripartirà; Joe fa ben attenzione a non commettere stupidi errori quando lavora coi suoi programmi.

Appena la lettera A è stata presentata dalla tastiera, il programma di Joe inserisce il lettore di nastro perché possa essere letto il primo nome e indirizzo, che (come ricorderete) è memorizzato sul nastro in carta, subito dopo il programma.

Il programma di Joe legge ora questo nome e indirizzo dal nastro perforato e lo immagazzina, sotto forma di dati nella memoria.

Poi il programma di Joe procede nel suo controllo sulla telescrivente finché lo spazio della cifra da pagare è direttamente sotto l'elemento stampante.

Il programma fa aspettare il microcomputer finché Joe non abbia battuto la cifra da pagare, prima in lettere poi in numeri.

TASTIERE

Vediamo un po' cosa fa il microcomputer ogni volta che Joe batte un carattere sulla tastiera della teletype.

Ricordate che, ogniqualvolta ci si riferisca al microcomputer, Joe è una vera lumaca. Il ritmo più veloce cui Joe può battere è di 3 caratteri al secondo.

Un microcalcolatore tipico esegue un'istruzione ogni 5 microsecondi, il che significa che esso esegue 200.000 istruzioni al secondo.

$$5\ \mu\text{sec} = 5 \text{ microsecondi} = \frac{5}{1.000.000} \text{ secondi}$$

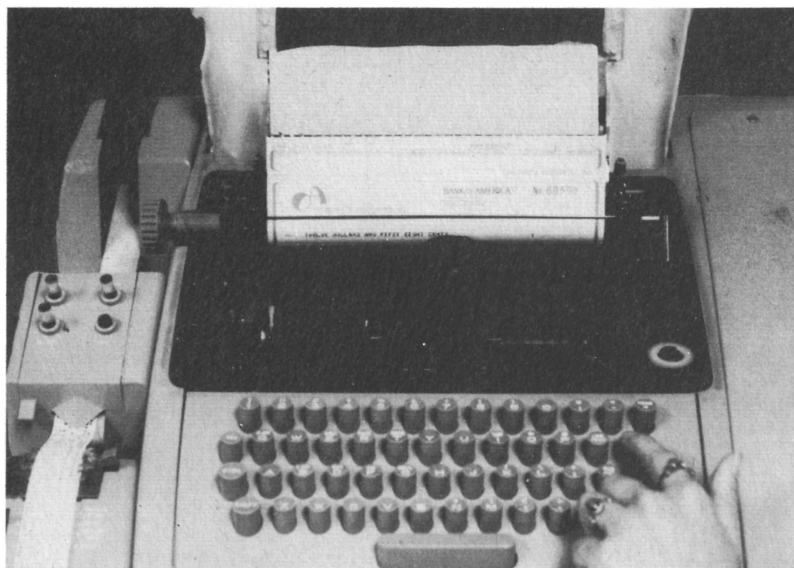
$$\text{istruzioni al secondo} = \frac{1.000.000}{5} = 200.000$$

Poiché Joe può battere 3 caratteri al secondo:

$$\text{istruzioni per battuta} = \frac{200.000}{3} = 67.777$$

Il microcomputer può quindi eseguire mediamente 67.777 istruzioni fra ogni battuta di tasto.
Chiaramente, il programma di Joe ha tempo di fare ben più che leggere il carattere in arrivo e riecheggiarlo — cosa che richiede meno di 20 istruzioni.

RECUPERO ERRORI

Visto tutto questo avanzo di tempo, **Joe decide di usare la tastiera per aiutarlo a venir fuori dai suoi guai,** poiché se può capitare un errore, è certamente lui a farlo. **Joe usa la tastiera anche per controllare il suo programma.**
Appena ricevuto un carattere dalla tastiera, il programma di Joe controlla per vedere che il carattere non sia un "ESCAPE" (errore: c'è un tasto apposito sulla tastiera della telescrivente).

RIPARTENZA

Dopo aver rivelato un ESCAPE, il programma avanza i controlli nella stampante fino all'inizio del secondo controllo; ritorna la stampante all'inizio della linea, quindi, **ritorna a maneggiare il solito nome e indirizzo.**

Così ogni volta che Joe fa un errore, egli può semplicemente battere l'ESCAPE e ripartire. Poi, il programma di Joe controlla se vi è il carattere di RITORNO CARRELLO.

Dopo aver rivelato un carattere di ritorno carrello, il programma di Joe legge il nome e indirizzo che sono ora in memoria, e li stampa sull'assegno.

L'indirizzo sarà prelevato dalla memoria ad opera del microcomputer, il quale farà si che l'indirizzo venga battuto nella zona appropriata dell'assegno.

IMSAI 8080

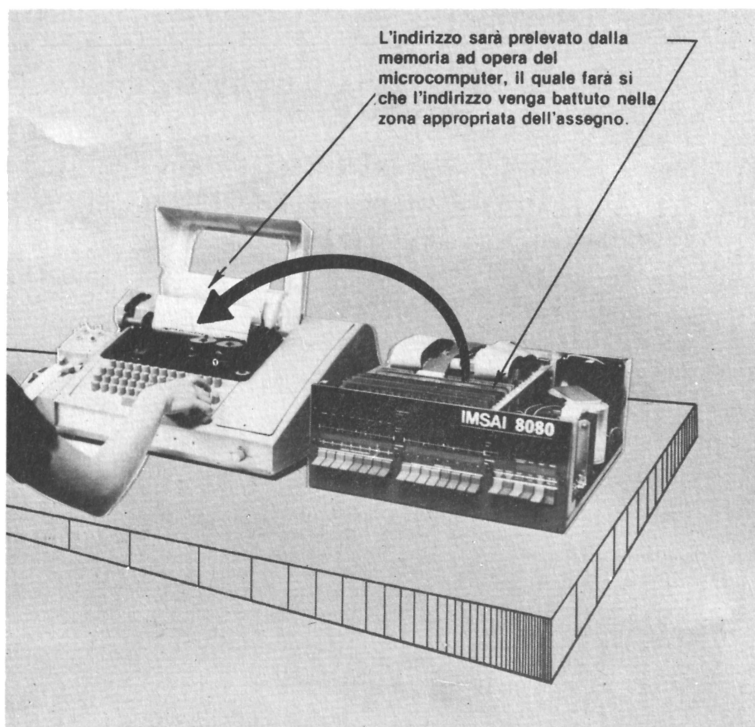



DIAGRAMMA DI FLUSSO DELLA LOGICA DI PROGRAMMA E SUOI SIMBOLI

Nell'intento di aiutarvi a capire la logica del programma di Joe, senza dover capire come Joe ha scritto il programma, esaminiamo la seguente figura, che rappresenta il diagramma di flusso della logica di programma. Le varie figure geometriche fanno parte della simbologia standard di uso generale la cui serie completa verrà data in una tabella in appendice.

START

Leggere il nome e indirizzo successivi dal nastro perforato; immetterli in memoria

Avanzare l'assegno nella stampante fino alla linea della cifra.

Ricevere il successivo input di un carattere dalla tastiera.

È un errore? Yes / No

È un ritorno carrello? Yes / No

Mandare in eco il carattere alla stampante

Avanzare l'assegno sulla stampante fino alla riga "pagherete a..."

Stampare nome e indirizzo che sono attualmente in memoria.

C'è ora un altro nome e indirizzo? Yes / No

STOP

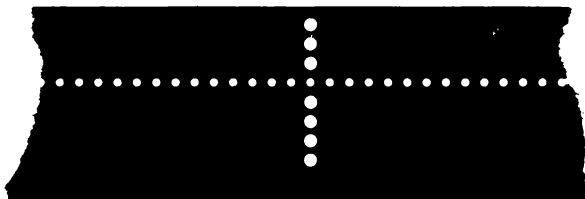
Figura 2-1. Diagramma di flusso per il programma Joe's Bill Paying

Se fate un passo indietro ed osservate il programma di Joe, esso consiste di una sequenza di passi complessivi, dove ogni nome e indirizzo viene letto ed un assegno è stampato durante ciascuno di questi passi. **E ciò è quanto accade durante un passo complessivo:**

1) il microcomputer posiziona l'assegno perché Joe possa battere la cifra;
2) Joe batte la cifra;
3) Joe batte un ritorno di carrello a fine cifra: il microcomputer automaticamente esegue il ritorno di carrello, quindi batte il nome e indirizzo del creditore;
4) il microcomputer legge dal nastro il successivo nome e indirizzo, ed avanza il rotolo di assegni fino alla linea della cifra dell'assegno successivo.

Se a Joe capitasse mai di commettere un errore, egli preme il tasto relativo (ESCAPE); questo gli permette di ripartire con l'assegno sul quale egli ha commesso tale errore. Come fa il programma di Joe a sapere quando non ci sono più nomi e indirizzi?

Il programma di lettura di nastro di Joe controlla anche i caratteri in arrivo. Joe sceglie una fila di 8 fori come carattere speciale per il "fine dati".

Come rileva la presenza di una fila di 8 fori, il programma intende che tutti i nomi e indirizzi sono stati letti, perciò si ferma.

A questo punto Joe ha pronti una serie di assegni che può firmare ed inserire in una busta con finestra trasparente, senza dover scrivere altri nomi e indirizzi; Joe ha risparmiato un sacco di tempo.

ALCUNE APPLICAZIONI DEI MICROCOMPUTER

Il programma di pagamenti di Joe è appena un semplice esempio del modo in cui voi potreste usare un microcomputer; ma esso è servito a dimostrare la maniera in cui le varie parti di un sistema microcalcolatore interagiscono mentre si esegue un programma.

In effetti, nonostante che Joe abbia un gran numero di conti da saldare, probabilmente sarebbe stato più veloce pagarli "a mano"; infatti la programmazione del microcomputer per fornire questa prestazione non farà risparmiare alcun tempo.

Abbiamo usato il programma di pagamenti come esempio non per mostrarvi perché un sistema a microcomputer sia una cosa economica da acquistare, ma piuttosto per mostrarvi come un microcomputer venga fatto lavorare.

Ma anche se Joe, che potrà aver da pagare 10 o 15 conti al mese, non riuscirà certamente a giustificare il pagamento di questi conti mediante computer, lo stesso programma che Joe ha scritto potrebbe maneggiare centinaia di conti così facilmente come maneggia i 10 ÷ 15 di Joe; e con centinaia di conti da pagare, il sistema microcalcolatore comincerebbe a risparmiare un sacco di tempo rispetto a quello necessario per fare lo stesso lavoro a mano.

Ma c'è un punto ancor più importante degno di nota; semplicemente cambiando il nastro di carta, Joe può far fare al suo intero sistema microcomputer un lavoro completamente diverso.

Così, se il microcomputer, non risulta economico per fare un lavoro, può diventarlo facendone 10. Supponiamo, per esempio, che Joe abbia speso 1 milione per il suo sistema microcomputer.

Un milione più un programma su nastro perforato generano un sistema microcomputer che costa 1 milione per un lavoro. Ma supponiamo che Joe abbia 10 diversi programmi, ognuno col suo nastro perforato. Estendendo il milione a tutti e 10 i lavori, egli dedurrà che sta spendendo solo 100.000 lire per programma.

Come Joe riesce a trovare altri utilizzi, il costo effettivo continua a scendere. E questo è ciò che rende i sistemi microcomputer così popolari: essi possono fare

qualunque cosa che possa essere definita mediante un programma.

L'unica limitazione esistente sul numero di programmi risiede nel tempo impiegato dal nostro microcomputer ad eseguirli tutti. **Joe ha molte altre applicazioni in mente per il suo sistema microcomputer.** Oltre a pagare i suoi conti, egli può mantenere aggiornata la sua rubrica di indirizzi e il saldo del suo libretto di assegni, tanto per citare alcuni altri esempi di tipo amministrativo.

Ma ci sono molte altre cose, non di tipo amministrativo, che il microcomputer può fare; per esempio, esso può giocare. Un sistema microcomputer può essere usato per fare semplici giochi sfruttando il vostro televisore, o può anche essere programmato per giochi più complessi, tipo gli scacchi.

In fondo alla mente di Joe c'è poi un'applicazione ancor più interessante per il suo microcomputer; la musica sintetizzata; scrivendo programmi che creano suoni si possono infatti pilotare direttamente degli altoparlanti.

Quando Joe avrà capito un po' meglio il suo microcomputer, ne vedremo delle belle.

Capitolo 3
COMPONENTI DEI SISTEMI A MICROCOMPUTER

QUELLO CHE SI VEDE NON È SEMPRE QUELLO CHE SI OTTIENE

Un giorno a Joe capita un evento sfortunato: il suo amico desidera gli venga restituito il terminale teletype.

Tornare al microcomputer senza occhi ed orecchi è fuori questione. Non solo quello del microcomputer è un prezzo troppo alto da pagare, solo per vedere luci che si accendono e si spengono, ma oltretutto Joe ha investito molto del suo tempo scrivendo programmi che gli sono utili.

E' così che Joe stringe la cinghia, afferra il libretto degli assegni e torna al grande magazzino computer. **Joe desidera insomma espandere il suo sistema microcomputer.**

Le opzioni che si presentano a Joe sono tali da confondergli la mente; e probabilmente confonderanno anche ciascuno di voi, finché non si sarà capito esattamente quello che si sta cercando: solo allora un po' di ordine comincerà a comparire dal caos.

Quando si mette assieme un sistema microcomputer, si devono prima di tutto scegliere le funzioni che il microcomputer deve svolgere: per esempio, il ricevere dati all'ingresso, lo stampare i risultati e l'immagazzinare informazioni, sono tutte "funzioni".

Successivamente occorre scegliere fisicamente l'apparecchio che serve (o che ci si può permettere) allo scopo di mettere in pratica ciascuna esecuzione delle funzioni che sono state scelte; per esempio, le informazioni possono essere memorizzate su nastro di carta, in cassette o su floppy-disk.

Ogniqualvolta si sceglie una particolare unità fisica, occorre decidere anche sulle opzioni e prestazioni addizionali di cui ci si prepara a sostenere il costo successivo.

Passiamo ora ad identificare i componenti che voi potete acquistare per il vostro sistema microcomputer, le funzioni cui essi adempiono all'interno dello stesso, e le opzioni comunemente disponibili.

Cominceremo con l'esaminare le funzioni effettuate ed i componenti che si possono concretamente acquistare allo scopo di ottenere ciascuna di quelle funzioni.

UNITA' FISICHE E LOGICHE NEI SISTEMI MICROCOMPUTER

Ritornando un attimo al primo capitolo, nella figura di pag. 1-2 si vedono illustrati i seguenti componenti:

1) il microcomputer vero e proprio
2) una tastiera
3) un display video
4) una stampante
5) una memoria informazioni di massa

Ricordate come nello stesso primo capitolo, abbiamo descritto "record" e "file" sia come entità fisica che come entità logica?

Il record o il file "fisici" sono la forma nella quale l'informazione è concretamente immagazzinata nel dispositivo di memoria, mentre un record o un file "logici" sono il modo nel quale voi, come utente di microcomputer, visualizzate e usate l'informazione memorizzata.

Possiamo estendere questo concetto ad un gradino ulteriore, prendendo i cinque componenti di un sistema a microcomputer elencati poco sopra, ed esaminando le funzioni che essi esplicano.

UNITA' LOGICA "IMMISSIONE DI INFORMAZIONE"

La tastiera diventa un'unità logica di "immissione dell'informazione", che però non è obbligatorio sia una tastiera; potrebbe essere il quadro comandi del microcomputer, o ogni altra parte del sistema accessorio capace di leggere informazioni, per esempio un lettore di nastro perforato.

UNITA' LOGICA "OPERATORE DI MESSAGGIO"

Il display video diventa un'unità logica "operatore di messaggio"; può consistere in un vero e proprio display video, una stampante da telescrivente o da macchina da scrivere, o comunque ogni altra attrezzatura accessoria capace di mettere a disposizione dei messaggi in forma comprensibile ai sensi umani.

UNITA' LOGICA USCITA RISULTATI

La stampante diventa un'unità logica per "uscita risultati". Già abbiamo visto come i risultati siano generati da una macchina dattilografica, da una telescrivente o da una stampante autonoma; ma ci sono molte altre maniere in cui vi può essere utile generare i vostri risultati.
Per esempio, potreste scrivere i risultati sul nastro di una cassetta, con l'idea di stamparli più tardi, quando il microcomputer sia scarico da altri lavori.

UNITA' LOGICA MEMORIA DI MASSA DELLE INFORMAZIONI

A suo tempo abbiamo visto tre soluzioni, per concretizzazare l'unità logica "memoria di massa per le informazioni": le unità rispettivamente a floppy disk, a nastro-cassetta e a nastro di carta perforato. Abbiamo pure esaminato le unità a dischi rigidi come alternativa a floppy.

Un dispositivo di memoria dati non sempre viene trattato come unità logica singola. Talvolta singoli "file" possono diventare unità logiche individuali, oppure gruppi di "file" logici possono costituire un'unità logica.

Talché un'unità logica può trasformarsi in molte unità logiche, così come, d'altro lato, più di un'unità fisica può costituire una singola unità logica.

Qualsiasi relazione unità fisica/unità logica che sia fattibile è anche possibile; e ciò può essere illustrato come segue:

"LU" = "logical unit"

Unità a floppy disk con cinque "file" fisici registrati e vari numeri di record su ciascun "file"

A

LU1

L'utente A tratta l'unità floppy disk come una singola unità logica di memoria di massa che contiene cinque "file" logici

B

LU1

LU2

L'utente B tratta l'unità floppy disk come due unità memoria di massa una contiene 3 "file" logici, l'altra ne contiene 2

C

LU1

LU5

LU2

LU4

LU3

L'utente C tratta ciascun "file" individuale come un'unità logica separata

D

L'utente D ha due unità floppy disk; il tutto fa parte di una singola unità logica di memoria di massa

Solamente il microcomputer vero e proprio non ha sostituti. Perciò, e solo in questo esempio che unità fisiche e logiche possono essere sempre la stessa ed unica cosa.

Esaminiamo alcune illustrazioni che definiscano più chiaramente le relazioni fra unità logiche e fisiche.

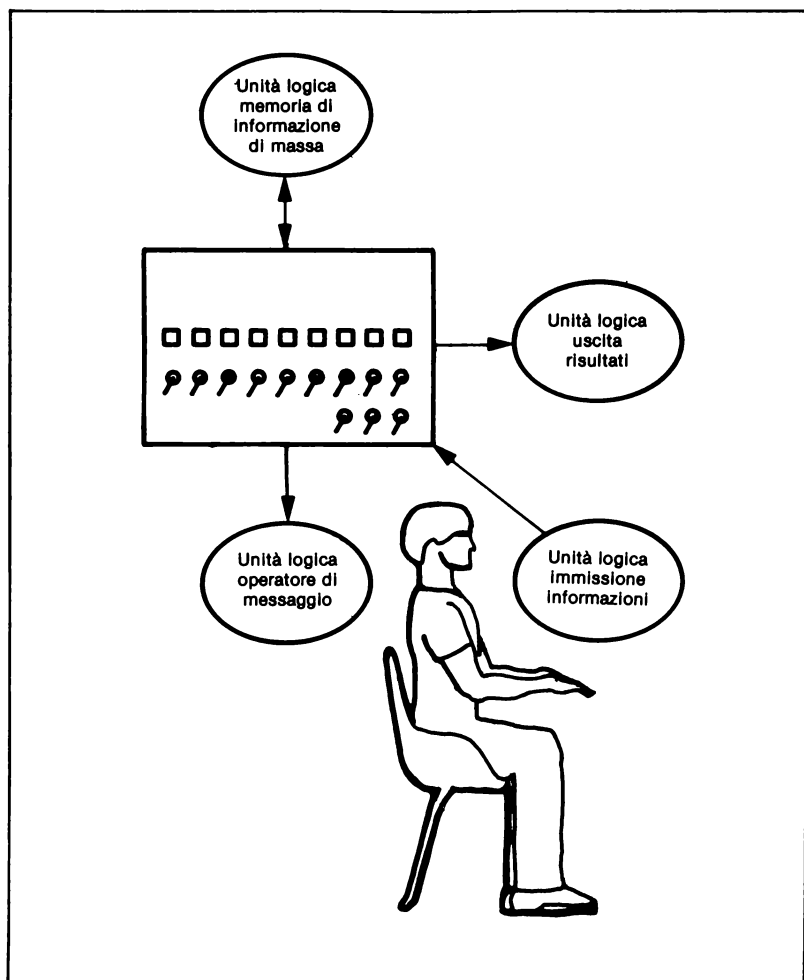



Figura 3–1. Unità logiche che circondano un microcomputer

Prima di tutto, la fig. 3-1 illustra le quattro unità logiche che circondano il microcomputer.

In fig. 3-2 queste quattro unità logiche sono mostrate sovrapposte alla configurazione del sistema microcomputer quale è stata inizialmente introdotto nel primo capitolo.

La fig. 3-3 illustra in quale relazione sono le unità logiche e fisiche nel semplice sistema microcalcolatore di Joe Bitburger, che consiste nel microcomputer e nel terminale telescrivente.

Mentre il concetto di unità logiche e unità fisiche può sembrare astruso, in realtà ne possiamo trovare molti paralleli nella nostra vita di tutti i giorni.

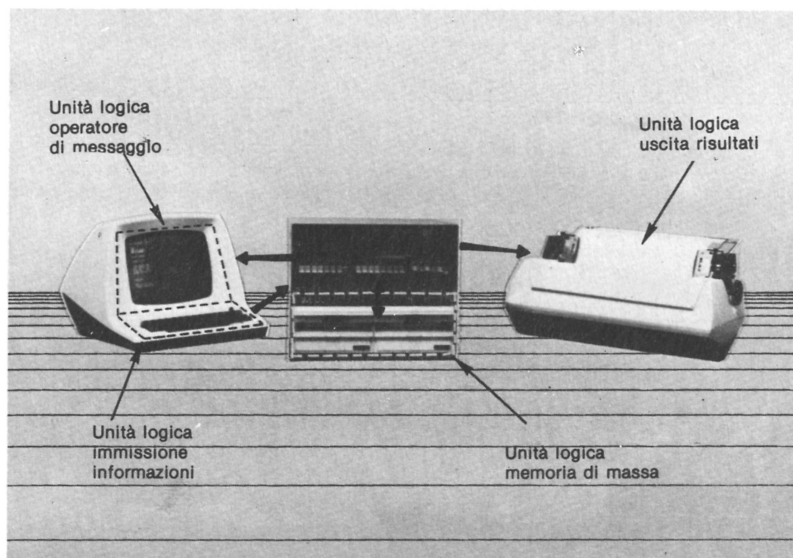



Figura 3-2. Unità logiche identificate per il sistema microcomputer del cap. 1.

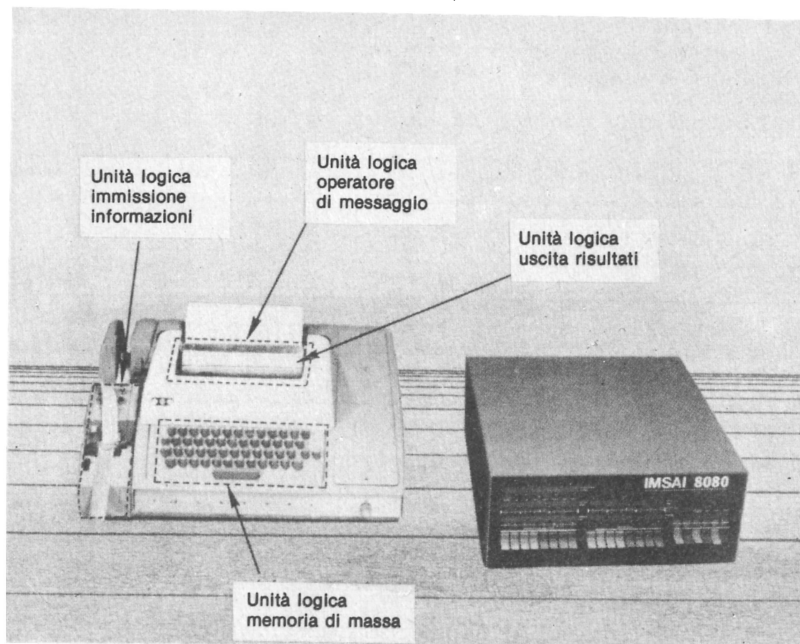



Figura 3-3. Unità logiche identificate per un terminale teletype.

Joe Bitburger ha bisogno, al mattino, di un'unità logica di "sveglia" per uscire dal letto in tempo; in tal caso un'unità fisica "orologio con allarme" completa l'unità logica di "sveglia". Un giorno, l'orologio con sveglia si guasta. Fortunatamente, ciò succede nella stagione in cui l'orologio suona quando sorge il sole; cosicché Joe lascia aperte le tapparelle, ed in tal modo usa il sole nascente attraverso la sua finestra aperta come l'unità fisica che concretizza la sua unità logica di sveglia.

Certo che se Joe avesse programmato se stesso a rispondere, al mattino, solo al trillo di un campanello, la commutazione con la luce che entra dalla finestra non avrebbe funzionato; ma Joe ha programmato se stesso, intelligentemente, in modo da rispondere ad ogni tipo di stimolo: perciò non ha problemi a commutare il suo stimolo di sveglia dal trillo della suoneria ai raggi del sole che entrano dalla finestra.

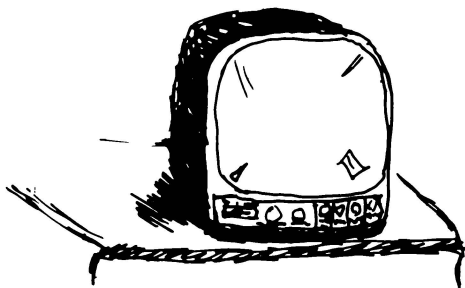
Joe normalmente per colazione prende una tazza di caffé; per questo scopo, Joe ha bisogno di un'unità logica "scalda acqua". L'unità logica "scalda acqua" può essere realizzata da un'unità fisica "pentolino"; ma quando il pentolino denuncia una frattura che fa perdere acqua, Joe lo sostituisce con un'altra unità fisica "vasetto per salsa".
Dopo la colazione, Joe deve cimentarsi con l'unità logica "trasporto al lavoro"; normalmente l'unità fisica che realizza l'unità logica di Joe "trasporto al lavoro" è l'autobus.

Un certo giorno all'autobus capita un incidente, cosicché Joe, per andare al lavoro, è costretto a riesumare la bicicletta; ora è la bicicletta che diventa l'unità fisica che concretizza l'unità logica "trasporto al lavoro".
Se Joe avesse programmato la sua vita secondo un preciso orario di autobus, ora avrebbe qualche problema; ma Joe si è programmato in modo da prendere in considerazione diversi mezzi per andare al lavoro, e da averne il tempo sufficiente.

Questi sono appena tre esempi di analogie fra unità logiche e fisiche che noi possiamo incontrare nella vita di tutti i giorni.

COMPONENTI ACCESSORI PER IL MICROCOMPUTER

Esaminiamo ora alcune delle combinazioni di unità fisiche reperibili in commercio.
Partiamo dalle combinazioni tastiera/display video.

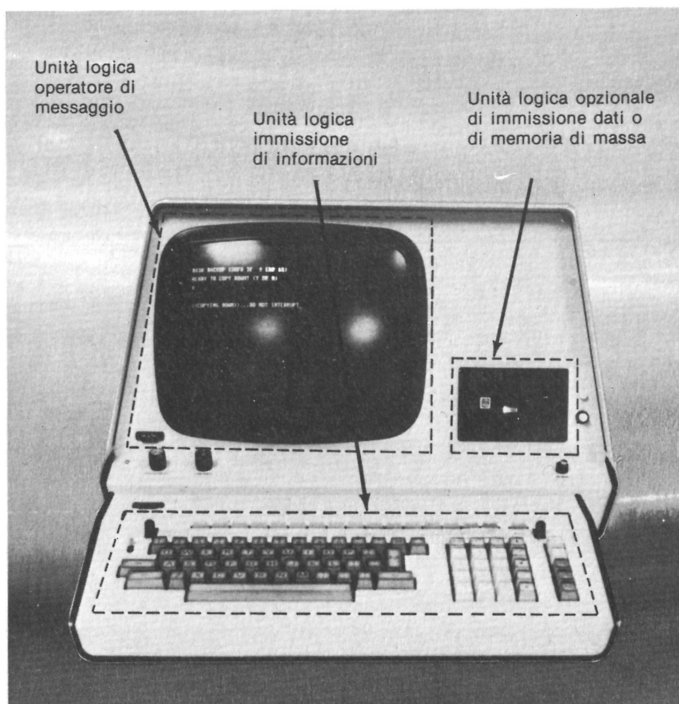
Tale combinazione è talmente comune che molta gente crede che le due unità debbano per forza stare assieme; ma non è obbligatorio, tant'è vero che si può acquistare un video display anche da solo, o anche usare il proprio televisore.

Pure la tastiera può essere acquistata separatamente.

Unità logica
immissione informazioni

Esistono unità tastiera/video display che comprendono anche un riproduttore di cassette; questo giranastri può servire come una sola parte, o costituire il tutto dell'unità logica "memoria di massa".

Unità logica operatore di messaggio

Unità logica immissione di informazioni

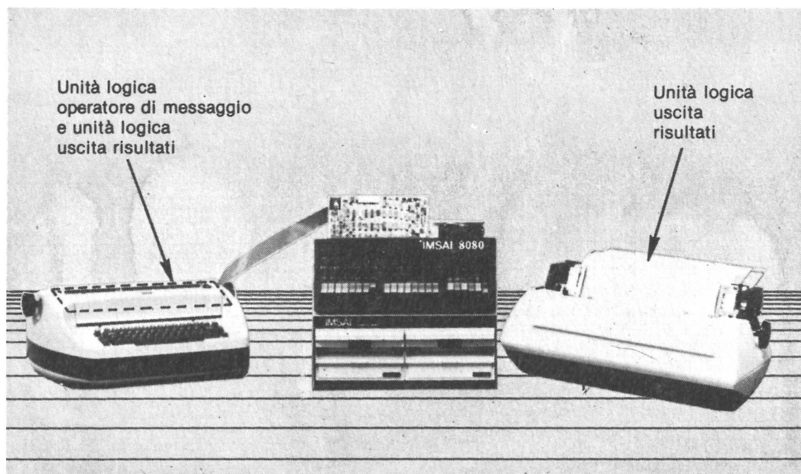
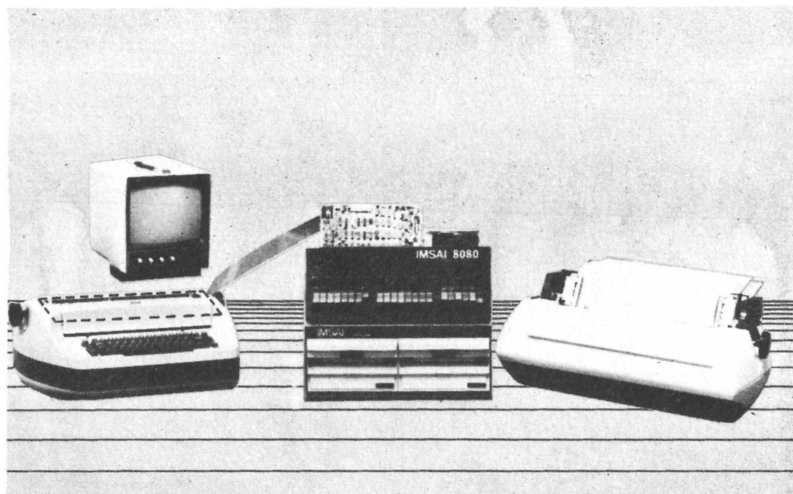
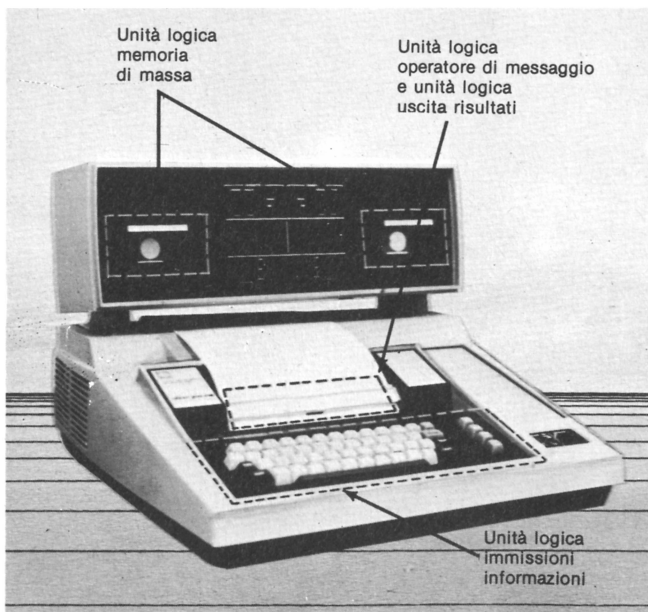
Unità logica opzionale di immissione dati o di memoria di massa

Si può anche acquistare una tastiera di quelle che già hanno l'aspetto di una parte integrale di microcomputer.

Il microcomputer è qui

Sol Terminal Computer

Unità logica immissione di informazioni

Anche le stampanti possono creare qualche confusione. Talvolta, l'unità logica "operatore di messaggio" e "uscita risultati" diventano una stessa unità fisica.

Ma potete anche prendere il medesimo sistema microcomputer, aggiungere una stampante separata, ed ora l'unità logica "uscita risultati" e quella "operatore di messaggio", quantunque siano ambedue stampanti, sono unità fisiche separate.

Aggiungete un video display, e potrete usare sia la macchina da scrivere che la stampante come unità logica "uscita risultati", mentre la macchina da scrivere o il video display servono come unità logica "operatore di messaggio".

Esiste una serie molto popolare di terminali, i Silent 700 della Texas I, una versione dei quali combina assieme tastiera, stampante e giranastri. Questo terminale può servire come unità logiche per immissione di informazione, operatore di messaggio, uscita risultati e memoria di massa.

RIDESIGNAZIONE DELLE UNITA' LOGICHE

RISULTATI CONSERVATI SU CASSETTA

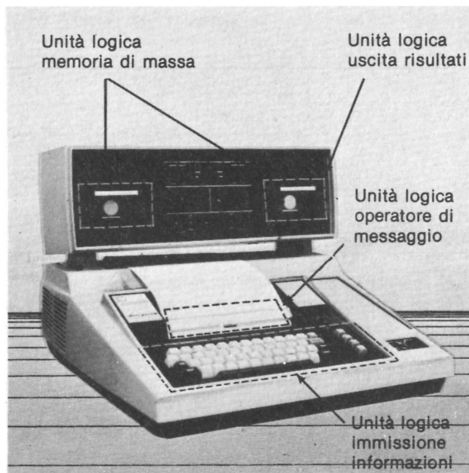
Usando il terminale ora nominato, può essere presa in considerazione un'altra interessante possibilità; supponete di avere dei risultati che desiderate stampare, ma di tali risultati non avete bisogno subito o non potete stare ad attendere la disponibilità della stampante.

Esiste allora una facile soluzione. Designate uno dei riproduttori di cassette come unità fisica corrispondente all'unità logica "uscita risultati".

Unità logica memoria di massa

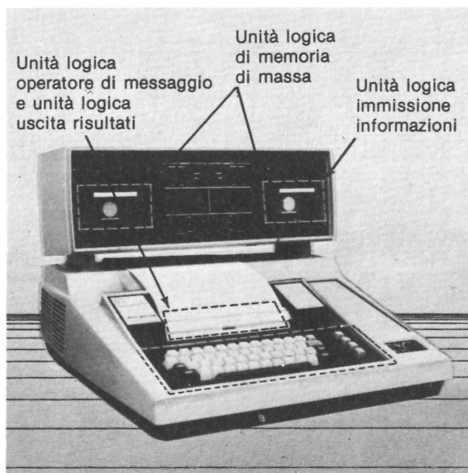
Unità logica uscita risultati

Unità logica operatore di messaggio

Unità logica immissione informazioni

Ora i vostri risultati verranno scritti molto rapidamente in una cassetta. **Estraete la cassetta e conservatela fino a fine giornata.** Quando ricaricate la cassetta **ridesignate il riproduttore come l'unità fisica corrispondente all'unità logica "memoria di massa", e la stampante come unità logica "uscita risultati". I vostri risultati verranno stampati durante la vostra assenza.**

Unità logica di memoria di massa

Unità logica operatore di messaggio e unità logica uscita risultati

Unità logica immissione informazioni

GESTIONE DEI DISPOSITIVI ESTERNI

Il bello dei microcomputer e la loro versatilità. Una volta che siate in possesso delle necessarie parti di "hardware" tutto quello che serve e un piccolo programma allo scopo di sfruttare ciascuna unità fisica per rappresentare ogni unità logica che sia ragionevole dal punto di vista fisico.
Per esempio, designare un'unità fisica "stampante" come unità logica di "immissione dati" e irragionevole in quanto fisicamente impossibile; una stampante è solamente capace di ricevere dati, mentre l'unità logica d'immissione dati deve trasmetterli, i dati.

Voi potete designare ogni ragionevole unità fisica come qualsiasi unità logica disponendo di un appropriato programma che colleghi nel vostro sistema microcomputer i due tipi di unità.

Questi programmi sono definiti come programmi di gestione dei dispositivi esterni, ed illustrati in fig. 3-4.

Vale la pena perdere un po' di tempo ad osservare questa figura, in quanto incorpora un numero di idee e complessi che sono difficili da afferrare per un principiante.

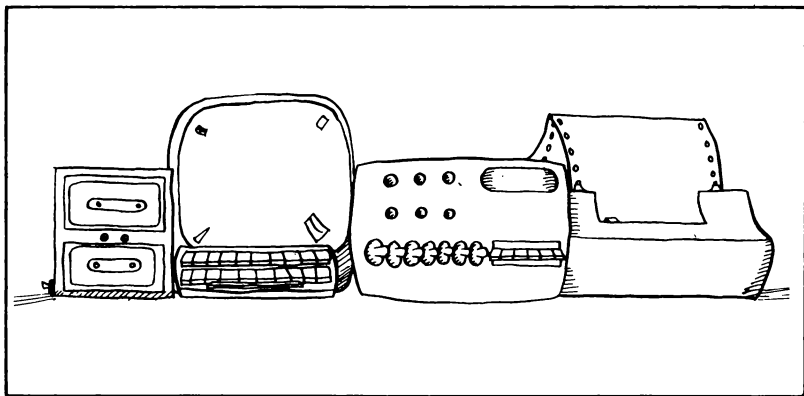
Le unità fisiche mostrate in fondo alla figura sono tutte attrezzature accessorie che voi potete vedere e toccare. Ciascuna di queste unità fisiche deve avere il proprio programma di gestione che provveda ai bisogni specifici dell'unità fisica. Un programma di gestione dispositivi esterni è semplicemente un programma di calcolatore, cioè una sequenza di numeri, come qualsiasi altro programma.
Quello che contraddistingue un programma di gestione da ogni altro è la logica del programma, il fatto che questa logica rifinisce le possibilità fisiche di un'unità fisica perché possa operare le funzioni richieste da un'unità logica.

Unità logica immissione informazioni
Unità logica operatore di messaggio
Unità logica uscita risultati
Unità logica memoria di massa
Queste sono le idee usate in fase di programmazione
Scegliere la logica fa parte del vostro programma
Gestione lettore nastro perforato
Gestione tastiera
Gestione stampante
Gestione display video
Gestione Floppy disk
Gestione unità a cassette
etc.
Questi sono i programmi del calcolatore
Lettore nastro carta
Tastiera
Stampante
Video display
Floppy disk
Unità a cassette
etc.
Queste sono le unità fisiche che potete vedere e toccare

Figura 3-4. Unità logiche e unità fisiche collegate usando un programma di gestione dispositivi esterni

I programmi di gestione dispositivi esterni a questo punto non vi sono più nuovi; ricordate che Joe Bitburger probabilmente è uscito ed ha comprato un programma di gestione (nel chip di una ROM) per controllare la sua teletype.

Un'unità logica non è niente di più che un'idea; non potrete mai né vedere né toccare un'unità logica. Per esempio, nel programma per il pagamento dei conti di Joe Bitburger, il lettore di nastro perforato diventa l'unità logica di magazzino dati, perché è quello che fornisce a Joe i nomi e gli indirizzi. Il concetto di informazioni immagazzinate, nel caso di Joe — nomi e indirizzi — è un'idea. L'idea è collegata ad una realtà fisica, un lettore di nastro, per mezzo del programma di gestione dispositivi esterni della teletype.

Supponiamo ora che Joe Bitburger abbia fatto un buon lavoro con lo scrivere il suo programma di pagamento conti. Ora che ha perso la sua teletype, supponiamo che la rimpiazzi con un terminale a display video, una tastiera, una stampante singola ed un paio di riproduttori per cassette.

Forse Joe dovrà far marcia indietro e riscrivere completamente il suo programma? Certamente no. **Tutto quello che Joe dovrà fare sarà rimpiazzare il programma di gestione per teletype con il programma di gestione per video display, tastiera, stampante e unità cassette.**

Così, semplicemente collegando le unità logiche alle unità fisiche nel nuovo modo corretto, tutto il suo sistema microcomputer lavorerà alla perfezione. Tutto ciò è illustrato in fig. 3-5.

In essa il programma pilota teletype di Joe è stato spezzato in quattro parti, e precisamente; un programma di gestione per tastiera di teletype, un programma di gestione per stampante di teletype, un programma di gestione per lettore di nastro perforato ed un programma di gestione per perforatore.

In realtà tutte e quattro le parti del programma pilota teletype risulteranno rinchiuse in un singolo chip di ROM, e l'intero blocco sarà trattato come un singolo programma di gestione di dispositivi esterni.

Joe deve togliere questo chip di ROM e sostituirlo con quattro nuovi programmi di gestione. I quattro nuovi programmi di gestione dei dispositivi esterni non sostituiscono i quattro preesistenti singolarmente uno per uno; infatti, come è illustrato, due nuovi programmi di gestione sostituiscono un singolo programma di gestione per stampante di teletype, mentre un programma di gestione per unità a cassette rimpiazza sia il programma di gestione di lettore di nastri per teletype sia quello del perforatore per teletype.

Programma gestione stampante video display

Programma gestione stampante teletype

Programma gestione schermo video display

Programma gestione stampante teletype

Programma gestione stampante

Programma gestione lettore di nastro teletype

Programma gestione unità a cassette

Programma gestione perforatore nastro teletype

Questi programmi sostituiscono questi programmi

Unità logica immissione informazioni

Unità logica operatore di messaggio

Unità logica uscita risultati

Unità logica memoria di massa

Il programma fondamentale di pagamento conti non cambia

Figura 3-5. L'uso dei programmi di gestione dispositivi esterni per rimpiazzare le unità fisiche.

Mentre i quattro programmi di gestione dispositivi per telescrivente potevano essere visti come un solo programma su una singola ROM, i quattro nuovi programmi si avviano così ad apparire come tre programmi distinti e separati:

1) i programmi di gestione per tastiera e schermo del video display;
2) il programma di gestione per la stampante;
3) il programma di gestione per l'unità a cassette.

Ciascun programma separato giustifica un'unità fisica separata; il singolo programma può essere disponibile come chip di ROM, cosa che normalmente non accade per tutti e tre i programmi.

Ciò avviene in quanto si possono acquistare unità fisiche separate e distinte in moltissime combinazioni diverse.

Supponiamo, per esempio, di avere tre differenti display video (A, B e C), tre differenti stampanti (P, Q e R) e tre diverse unità riproduttrici di cassette (X, Y e Z); serviranno 27 diverse chip di ROM per ottenere tutte le combinazioni dei programmi di gestione che queste nove unità fisiche possono richiedere in ciascuna combinazione. Ciò si può illustrare come segue:

ROM 1 A+P+X
ROM 2 A+P+Y
ROM 3 A+P+Z
ROM 4 A+Q+X
ROM 5 A+Q+Y
ROM 6 A+Q+Z

-
-
-

ROM 25 C+R+X
ROM 26 C+R+Y
ROM 27 C+R+Z

Occorre conoscere cosa sono i programmi di gestione, anche se probabilmente non vi servirà mai creare un programma di gestione.

Esattamente come Joe ha comprato un chip di ROM per i suoi programmi di gestione (o bootstrap)per la telescrivente, voi semplicemente acquisterete programmi di gestione sotto forma di unità preregistrate; infatti, fintanto non sarete dei programmatori di microcomputer molto esperti, non prenderete mai in considerazione lo scrivere i vostri personali programmi di gestione.

ALTERNATIVE NEI COMPONENTI DEI SISTEMI MICROCOMPUTER

Nei capitolo 1 e 2 sono state descritte le necessarie funzioni espletate da ciascuno dei componenti di un sistema microcomputer; ma non sono state discusse opzioni per i componenti.

Non è sempre possibile differenziare chiaramente una necessità da un'opzione, ma noi fisseremo che le informazioni sui sistemi a microcomputer dei capitoli 1 e 2 costituiscono la necessità, mentre quello che ora andremo a descrivere costituisce le opzioni.

ALTERNATIVE ALL'UNITA' DISPLAY VIDEO

Cominciamo ad esaminare le alternative che si possono reperire nel video display. La fig. 3-6 illustra la logica di un programma di gestione per controllare i più elementari terminali tastiera e display video.

Il programma in fig. 3-6 fa molto poco: aspetta una battuta della tastiera, e come la rinosce, la logica del programma si dirama in una delle tre seguenti direzioni:

1) visualizza un carattere visualizzabile
2) risponde ad un codice di controllo per il video display.
3) trasmette ciascun codice di controllo per il sistema microcomputer al microcomputer stesso.

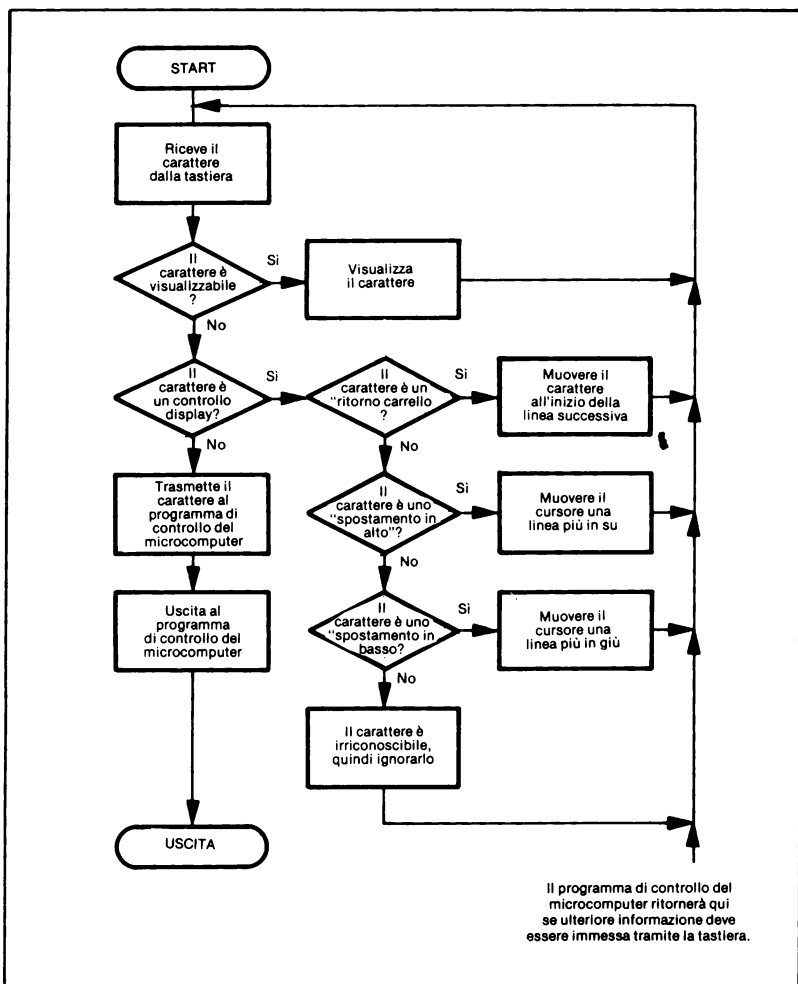



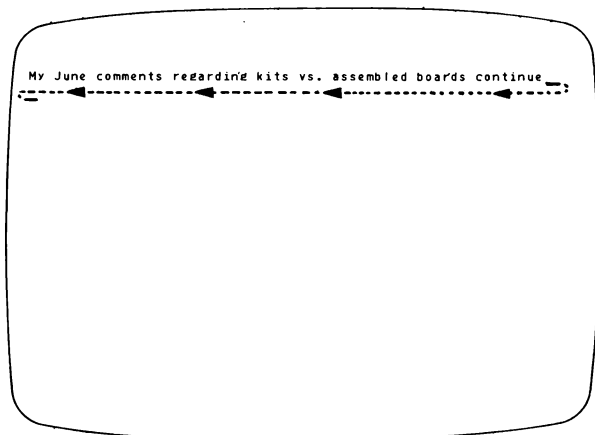
Figura 3-6. Grafico di flusso per un semplice programma di gestione per display video

Se la battuta consiste in una lettera dell'alfabeto, in un numero o in ogni altro carattere riproducibile, il programma fa scrivere il carattere sul display video; ricordiamo che ciò è indicato come eco.

La battuta può quindi costituire un controllo per il video display.

CURSORE SULLO SCHERMO

I controlli illustrati in fig. 3-6 includono un ritorno carrello muovendosi in alto o in basso di una linea. Poiché non c'è nessun meccanismo di stampa per dirvi dove siete sullo schermo del video, tutti i display usano quello che viene chiamato un "cursore", che è un piccolo punto, o quadratino, luminoso posizionato ove verrà visualizzato il carattere successivo.
Premendo quindi il tasto del ritorno carrello al terminale del video display, il cursore si sposta all'inizio della linea successiva inferiore.

```
My June comments regarding kits vs. assembled boards continue
```

Gli altri due controlli del video display illustrati nella logica di fig. 3-6 spostano il cursore di una linea verso l'alto

```
FROM THE FOUNTAINHEAD - SEPTEMBER 1977

My June comments regarding kits vs. assembled boards continue
to raise much controversy.  Bill Godbout of Godbout Electronics
called asking that his name be added to the list of suppliers
who deal only in tested parts.  After some discussion between
Bill and me we reached a consensus that there appear to be two
major types of company supplying hardware to hobbyists:

1)  There are companies such as Godbout Electronics, Newman
    Computer Exchange, E & L instruments, and for that matter,
    my own company, Osborne & Associates, that existed long before
    there was any hobby market.
```

o di una linea verso il basso

```
                FROM THE FOUNTAINHEAD - SEPTEMBER 1977

My June comments regarding kits vs. assembled boards continue
to raise much controversy.  Bill Godbout of Godbout Electronics
called asking that his name be added to the list of suppliers
who deal only in tested parts.  After some discussion between
Bill and me we reached a consensus that there appear to be two
major types of company supplying hardware to hobbyists:

1)  There are companies such as Godbout Electronics, Newman
    Computer Exchange, E & L instruments, and for that matter,
    my own company, Osborne & Associates, that existed long before
    there was any hobby market.
```

Com'è illustrato qui sopra, quanto del testo è riprodotto sullo schermo, potete modificarlo usando il cursore; ovunque gli capiti di essere, potete correggere qualsiasi errore nel vostro testo come segue:

```
                FROM THE FOUNTAINHEAD - SEPTEMBER 1977

My June comments regarding kits vs. assembled boards continue
to raise much controversy.  Bill Godbout of Godbout Electronics
called asking that his name be added to the list of suppliers
who deal only in tested parts.  After some discussion between
Bill and me we reached a consensus that there appear to be two
major types of company supplying hardware to hobbyists:

1)  There are companies such as Godbout Electronics, Newman
    Computer Exchange, E & L instruments, and for that matter,
    my own company, Osborne & Associates, that existed long before
    there was any hobby market.

2)  There are companies that cane into being specifically to serve
```

```
                FROM THE FOUNTAINHEAD - SEPTEMBER 1977

My June comments regarding kits vs. assembled boards continue
to raise much controversy.  Bill Godbout of Godbout Electronics
called asking that his name be added to the list of suppliers
who deal only in tested parts.  After some discussion between
Bill and me we reached a consensus that there appear to be two
major types of company supplying hardware to hobbyists:

1)  There are companies such as Godbout Electronics, Newman
    Computer Exchange, E & L instruments, and for that matter,
    my own company, Osborne & Associates, that existed long before
    there was any hobby market.

2)  There are companies that came into being specifically to serve
```

Un certo numero di controllo del video display, anche se non indicati nella logica di fig. 3-6, costituiscono tuttavia alternative comuni; essi includono:

1) **spaziatura in avanti;** tenendo abbassata la barra di spaziatura, il cursore procederà verso l'estremo della linea.

```
FROM THE FOUNTAINHEAD - SEPTEMBER 1977

My June comments regarding kits vs. assembled boards continue
to raise much controversy. Bill Godbout of Godbout Electronics
called asking that his name be added to the list of suppliers
who deal only in tested parts. After some discussion between
Bill and me we reached a consensus that there appear to be two
major types of company supplying hardware to hobbyists:

1)  There are companies such as Godbout Electronics, Newman
    Cpmputer Exchange, E & L instruments, and for that matter,
    my own company, Osborne & Associates, that existed long before
    there was any hobby market.
```

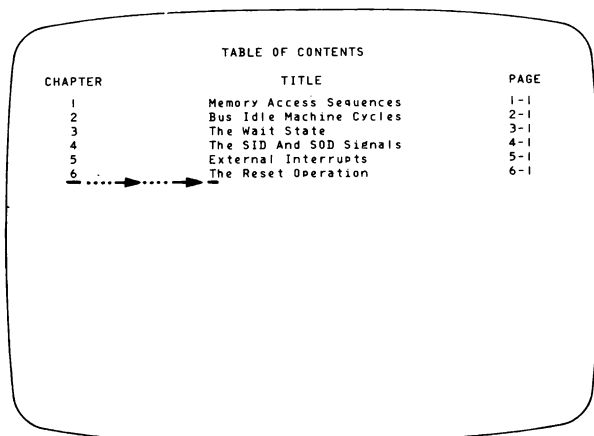
A seconda di come è stato progettato il vostro video, il cursore può arrestarsi quando raggiunge la fine di una linea, può spostarsi dalla fine della linea all'inizio della stessa, o può muoversi dalla fine all'inizio della linea successiva.

2) **spaziatura indietro;** premendo il tasto di arretramento, il cursore semplicemente si muove nella direzione opposta a quella illustrata nel caso precedente;

3) **tabulazione**; molti terminali a display video vi permettono di effettuare la tabulazione, dopo di che, premendo il tasto di tabulazione, il cursore salta alla successiva posizione di colonna all'interno della linea che si sta sviluppando sul video display.

```
                    TABLE OF CONTENTS

CHAPTER                 TITLE                        PAGE

   1           Memory Access Sequences               1-1
   2           Bus Idle Machine Cycles               2-1
   3           The Wait State                        3-1
   4           The SID And SOD Signals               4-1
   5           External Interrupts                   5-1
   6           The Reset Operation                   6-1
```

4) **Inserzione e cancellazione del testo**; i display video possiedono un vantaggio molto importante in confronto ad ogni stampante; si può infatti spostare il testo, e quindi molti display video approfittano di questa possibilità per inserire o cancellare del testo.

```
              FROM THE FOUNTAINHEAD - SEPTEMBER 1977

My June comments regarding kits vs. assembled boards continue
to raise much controversy. Bill Godbout of Godbout Electronics
called asking that his name be added to the list of suppliers
who deal only in tested parts. After some discussion between
Bill and me we reached a consensus that there appear to be two
major types of company supplying hardware to hobbyists:

1)  There are companies such as Godbout Electronics, Newman
    Computer Exchange, E & L instruments, and for that matter,
    my own company, Osborne & Associates, that existed long before
    there was any hobby market.

2)  There are companies that came into being specifically to serve
    the hobby market, once it had formed.

Companies that existed before the hobby market tend to buy only
tested parts, because that is what they had to do in order to
serve their prior industrial customer base. Many companies that
were formed specifically to service the hobby market tend to buy
untested parts, leaving it up to the kit buyer to test the parts
by trying to use them.
```

```
              FROM THE FOUNTAINHEAD - SEPTEMBER 1977

My June comments regarding kits vs. assembled boards continue
to raise much controversy. Bill Godbout of Godbout Electronics
called asking that his name be added to the list of suppliers
who deal only in tested parts. After some discussion between
Bill and me we reached a consensus that there appear to be two
major types of company supplying hardware to hobbyists:

1)  There are companies such as Godbout Electronics, Newman
    Computer Exchange, E & L instruments, and for that matter,
    my own company, Osborne & Associates, that existed long before
    there was any hobby market.

2)  There are companies that came into being specifically to serve
    the hobby market and no other market, once it had formed.

Companies that existed before the hobby market tend to buy only
tested parts, because that is what they had to do in order to
serve their prior industrial customer base. - Many companies that
were formed specifically to service the hobby market tend to buy
untested parts, leaving it up to the kit buyer to test the parts
by trying to use them.

I consider the discussion of kits vs. assembled boards, and tested
```

CONTROLLO DEL MICRO-COMPUTER DALLA TASTIERA

Un carattere inserito mediante la tastiera può corrispondere ad una funzione di controllo che non ha niente a che fare col video display. Ricordate che i programmi di Joe Bitburger contengono sempre un carattere per recupero di errore? Si tratta di un carattere che fa ripartire una parte del programma, o che in un modo o nell'altro permette a Joe di venir fuori dai guai. I terminali video avranno sempre uno o più di questi tasti, che però sono tasti speciali perché non rappresentano caratteri visualizzabili o controlli di schermo.

Alcuni terminali a video display riservano alcuni speciali controlli a compiti specifici; per esempio, può esservi un tasto per disconnettere il terminale dal microcomputer; oppure può esserci un altro tasto per fermare o far partire il microcomputer una volta che esso sia stato collegato al terminale.

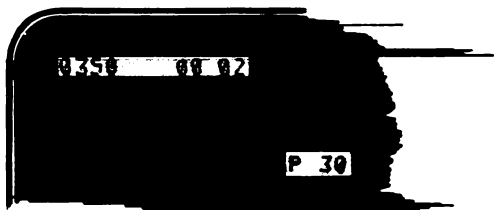
Se il microcomputer non ha pannello frontale, sono allora speciali caratteri di controllo inseriti da una tastiera che sostituiscono gli interruttori del pannello frontale.

Se invece il microcomputer ha un pannello comandi, spesso si presenta l'alternativa di usare una tastiera anziché detti comandi.

MINUSCOLE E MAIUSCOLE

I display con maiuscole e minuscole sono un'altra opzione possibile; si tratta naturalmente di terminali costosi, mentre quelli più semplici ed economici hanno solo caratteri maiuscoli. La tastiera possiede un apposito tasto, che permette di passare dalle lettere minuscole a quelle maiuscole, e viceversa.

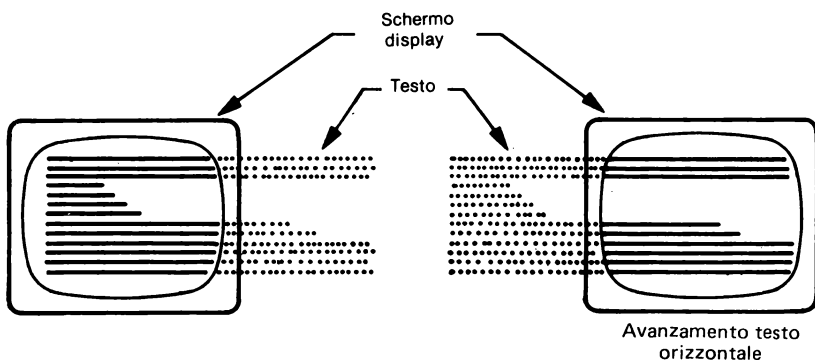
INVERSIONE DI DISPLAY

Alcuni video display permettono di rovesciare la luminosità dello schermo, in modo tale che i caratteri, in nero, sono rappresentati su un fondo bianco che prende in parte o del tutto lo schermo.

SCORRIMENTO ORIZZONTALE

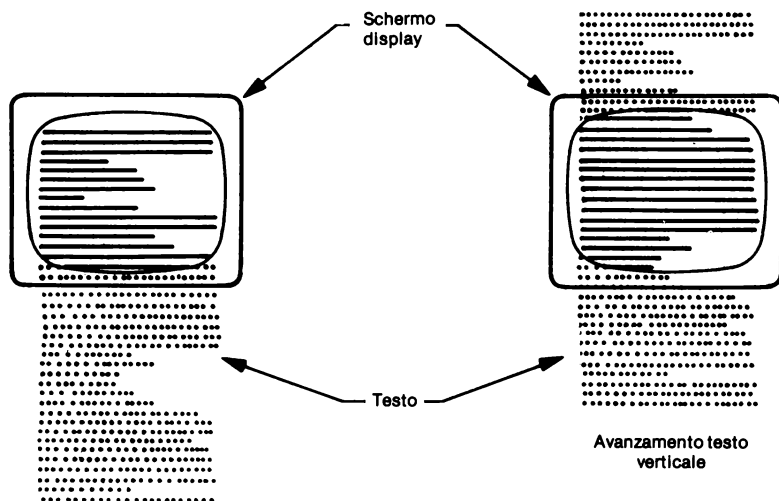
Alcuni display permettono di far scorrere l'immagine orizzontalmente o verticalmente. Lo scorrimento orizzontale permette di leggere linee di testo che siano più lunghe delle dimensioni dello schermo. In tal caso lo schermo si comporta come una piccola finestra sulle linee di testo, come in figura.

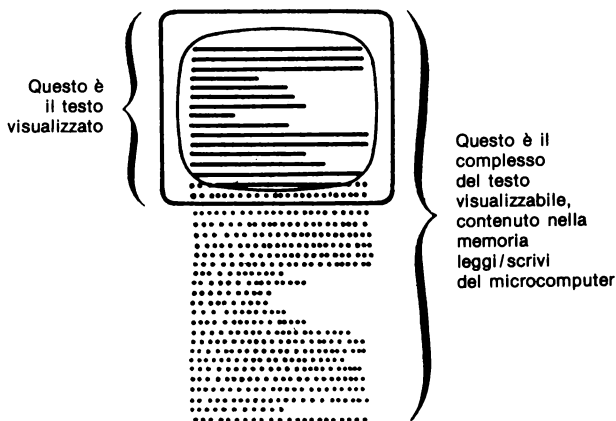
Schermo display

Testo

Avanzamento testo orizzontale

SCORRIMENTO VERTICALE

Lo scorrimento verticale è molto più comune di quello orizzontale. In questo caso, immaginate che il testo sia costituito da più linee di quelle che lo schermo può visualizzare; dovrete quindi poter far scorrere il testo verso l'alto e verso il basso.

Avanzamento testo verticale

Esistono due modi nei quali i terminali eseguono lo scorrimento verticale, e poiché uno è molto migliore dell'altro, li descriviamo ambedue.

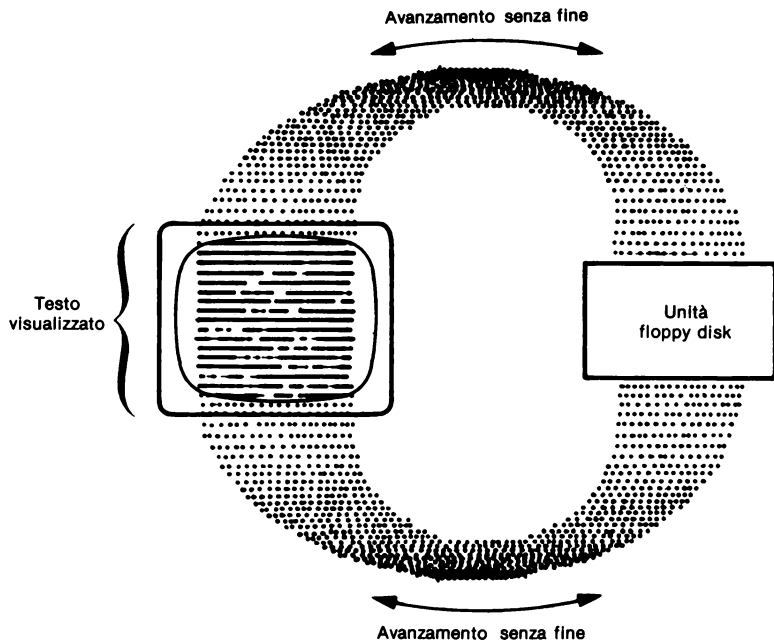
La versione meno consigliabile di avanzamento verticale conserva in memorie leggi/scrivi locali tutto il testo che può essere visualizzato; si veda per questo la figura seguente.

Se disponete di questo tipo di scorrimento, non potete far scorrere oltre l'inizio o sotto la fine del testo correntemente contenuto nella memoria leggi/scrivi. Se inserite del testo e superate la disponibilità della memoria, non farete altro che perdere informazioni dalla cima o dal fondo del testo.

Se il vostro sistema microcomputer ha un'unità a floppy disk, allora una ben progettata opzione di scorrimento collegherà l'unità floppy disk alla memoria leggi/scrivi entro la quale sta per essere immagazzinato il testo visualizzato. Ora, facendo scorrere in alto o in basso il testo nella memoria, i programmi all'interno del sistema microcomputer automaticamente memorizzeranno parte del testo della memoria leggi/scrivi nel floppy disk e porteranno del testo nuovo dal floppy alla memoria; il tutto vi apparirà come se il testo venga fatto scorrere indefinitamente. Il tutto può essere illustrato come segue.

Avanzamento senza fine

Testo visualizzato

Unità floppy disk

Avanzamento senza fine

Se inserite altro testo usando questa tecnica di scorrimento più sofisticata, e se superate la disponibilità di memoria leggi/scrivi del computer, allora il testo sovrabbondante verrà semplicemente scritto sul floppy disk e quindi non sarà perduto.

Da notare che queste opzioni di scorrimento in realtà non hanno nulla a che fare col terminale a display video o con la tastiera; si tratta di opzioni che si affidano all'intero sistema microcomputer ed al modo con cui esso è stato programmato.

DISPLAY GRAFICO

Il video display grafico è un'altra utile alternativa; esso consente di riprodurre illustrazioni oltre che caratteri. I terminali a display grafico economici riproducono in bianco e nero, ma non in grigio; essi quindi producono disegni, cioè linee rette, cerchi, punti e sagome piene. Invece i tipi più costosi di terminali a display grafico permettono di riprodurre tutto quello che può essere riprodotto da uno schermo televisivo, con migliori dettagli.

Esistono display video a colori così come ne esistono in bianco e nero, così come avviene per televisori.

Un display video a colori vi dà in più la possibilità di specificare il colore nel quale apparirà un carattere o un segno grafico.

PENNA LUMINOSA

Alcuni video display molto costosi permettono di scrivere sul display con una "penna luminosa". La logica elettronica che è oltre il display video "ricorda" i punti dello schermo che sono stati illuminati dal raggio della penna luminosa.

OPZIONI PER LA TASTIERA

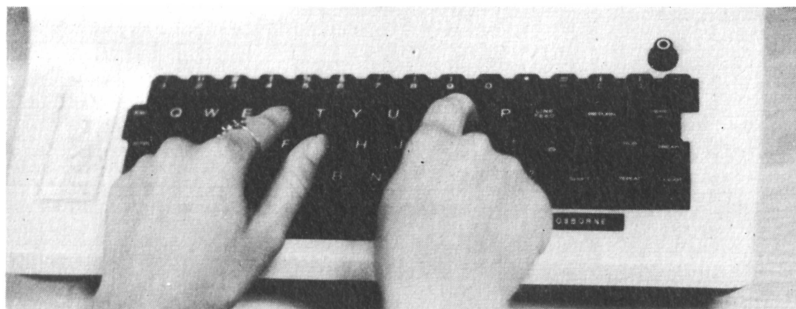
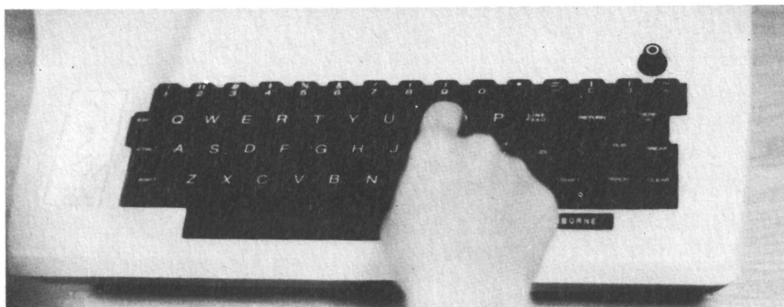
Esaminiamo ora alcune alternative di prestazione della tastiera.

Che cosa può dire a proposito della velocità con cui vengono battuti i tasti? Potrà, il microcomputer avere il tempo di processare un carattere prima che ne venga battuto un altro?
Nel caso del semplice programma di gestione illustrato in fig. 3-6, la risposta è quasi certamente sì.
Come è già stato discusso, i più veloci dattilografi del mondo effettuano (in media) nove battute al secondo, il che dà al vostro microcomputer il tempo di eseguire più di 20.000 istruzioni fra le singole battute.
Il programma di fig. 3-6 userà appena 100 ÷ 200 delle 20.000 istruzioni, e questo è tutto. Bisognerà però ricordare che nove battute al secondo rappresentano la cadenza media, non la massima cadenza delle battute.
E inoltre anche se il dattilografo più veloce eseguisse non più di nove battute al secondo, è sempre possibile premere due tasti quasi simultaneamente, dato che si usa battere con più di 1 dito.

Esistono due prestazioni opzionali che vi possono aiutare ad evitare problemi di questo tipo.

ROLLOVER

Parliamo prima di tutto del rollover. Vediamo cosa succede se voi scrivete così velocemente da premere un tasto;

poi batterne un altro

quindi lasciare il primo tasto.

L'istante nel quale viene registrata una battuta è molto importante. In qualsiasi normale macchina scrivente, ogni volta che voi premete un tasto, scrivete il carattere che gli corrisponde; se premete un secondo tasto quasi contemporaneamente come più sopra illustrato, una macchina scrivente elettrica provvederà a bloccare la seconda battuta, e non accadrà nulla. In una scrivente meccanica il secondo martelletto batterà sul retro del primo, e quindi il secondo carattere non verrà stampato.

Ma nel mondo dell'elettronica non c'è alcun bisogno di essere tanto restrittivi. Mentre vien premuto un tasto, la logica elettronica può rilevare qualsiasi altro tasto venga battuto, e può "ricordare" la seconda pressione di tasto intanto che non è stato stampato il primo carattere; tutto ciò può essere espresso come segue.

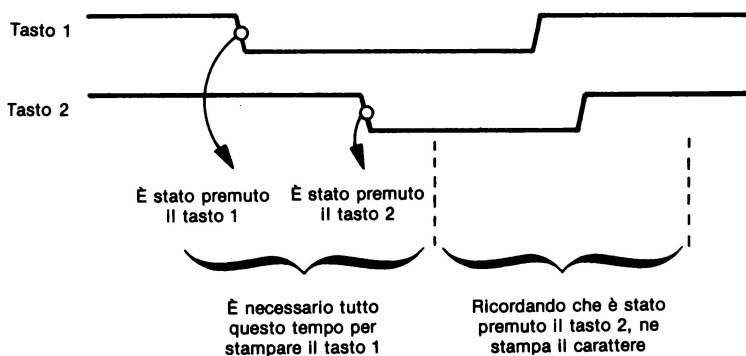
Tasto 1

Tasto 2

È stato premuto il tasto 1

È stato premuto il tasto 2

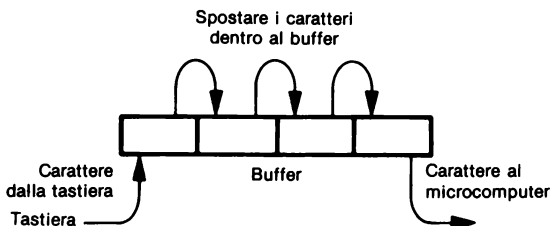
È necessario tutto questo tempo per stampare il tasto 1

Ricordando che è stato premuto il tasto 2, ne stampa il carattere

Il rollover è un'opzione molto consigliabile da avere disponibile in qualsiasi tastiera, poiché non ha niente a che fare con la velocità di esecuzione del computer.

Se avete dita agili, potrete frequentemente trovarvi voi stessi a sovrapporre battute; se la vostra tastiera non ha il rollover, ogni volta perderete la seconda battuta.

BUFFER

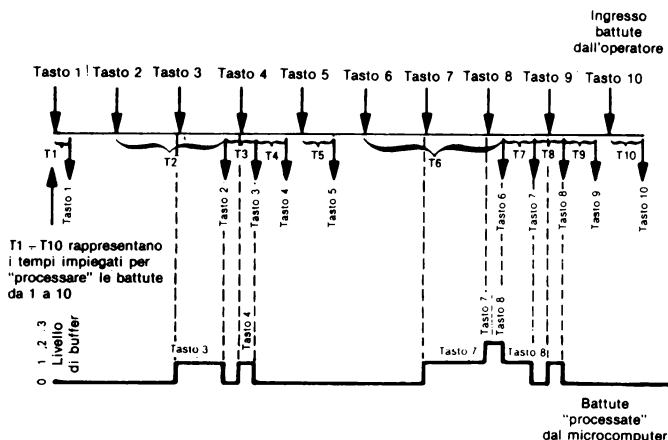
La seconda tecnica usata per rendere le tastiere a prova di errore consiste nell'avere a disposizione alcune posizioni di memoria entro le quali i codici del carattere vengono immagazzinati mentre aspettano di essere trasmessi al microcomputer.

Ciò si può illustrare come segue.

e la posizione di memoria è indicata come buffer (separatore-tampone).

Quantunque il buffer sia qui indicato con settori sufficienti per immagazzinare quattro caratteri, un buffer vero e proprio può contenere qualsiasi numero di caratteri compreso (normalmente) fra due e otto.

Un buffer dà al microcomputer più tempo per elaborare caratteri occasionali che richiedano tempi lunghi di risposta.

Ora, per il semplice programma di gestione tastiera di fig. 3-6, un buffer non servirebbe ad alcun possibile scopo utile. Ma non appena il programma di gestione diventa più complesso, è possibile che alcune battute richiedano più tempo del disponibile per essere processate; è ora che entra in gioco il buffer.

Se una battuta richiede in media più di 20.000 istruzioni da processare, il buffer verrà colmato e ne sarà superata la capacità, per quanto ampia possa essere. Ma se saranno solo alcune battute a richiedere più di 20.000 istruzioni da processare, allora il buffer lavorerà correttamente, secondo quanto qui riportato.

Esaminiamo questo grafico. Le battute sono illustrate in situazione di intervalli regolari di tempo (in alto).

Il carattere 1 è processato molto rapidamente durante l'intervallo di tempo T1; il buffer non è necessario.
Al contrario, il carattere 2 richiede un tempo di processo sostanzioso; infatti, prima che il carattere 2 sia stato completamente processato, viene inserito il carattere 3. In questo caso, il buffer deve conservare il codice di quest'ultimo carattere; e non appena è terminato il tempo di processo del carattere 2, il codice del 3 è estratto dal buffer ed esso pure processato; il buffer ora è vuoto. Ma la fase in cui è stato processato il carattere 3 è stata allungata dal carattere 2, cosicché il carattere 4 viene inserito prima che il 3 sia stato completato.
Il codice che rappresenta il tasto 4 viene quindi conservato nel buffer finché non è finito di processare il 3.
Il carattere 6 è la successiva battuta che richiede un sacco di tempo per essere processata; infatti i tasti 7 e 8 sono stati premuti ambedue prima che il 6 sia stato completamente processato. A questo punto, sono due i codici di carattere ad essere conservati nel buffer: il 7 e l'8.

Quando il carattere 6 è completamente processato, il codice del 7 è estratto dal buffer, lasciandovi solo quello dell'8.

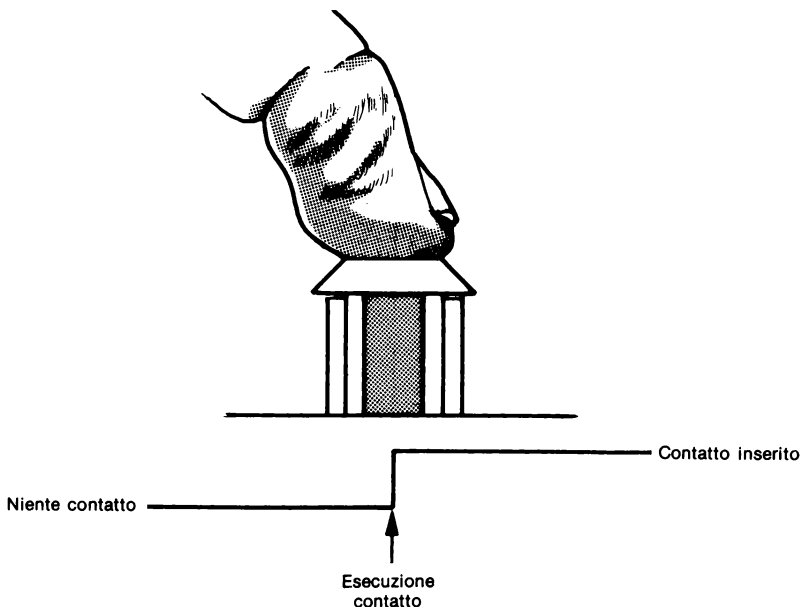
Il carattere 7 è completamente processato prima che venga inserito il 9, dopo di che il buffer resta vuoto. Ma il processo del carattere 9, essendo stato lui pure ritardato, fa sì che anche il codice di tale carattere vi sia immagazzinato per un breve periodo.

Non fosse stato per il buffer, i caratteri 3, 7 e 8 sarebbero andati persi. **La nostra discussione sulle tastiere multiple giunge ad aspetti non ovvi che vanno quindi considerati: esattamente, quand'è che viene rivelato che un tasto è stato premuto?**

Rispondere a questa domanda non è così semplice come può apparire. Si può semplicemente fissare che il premere un tasto provochi la chiusura di un qualche tipo di contatto, e in tale istante il tasto si possa considerare effettivamente premuto.

Contatto inserito

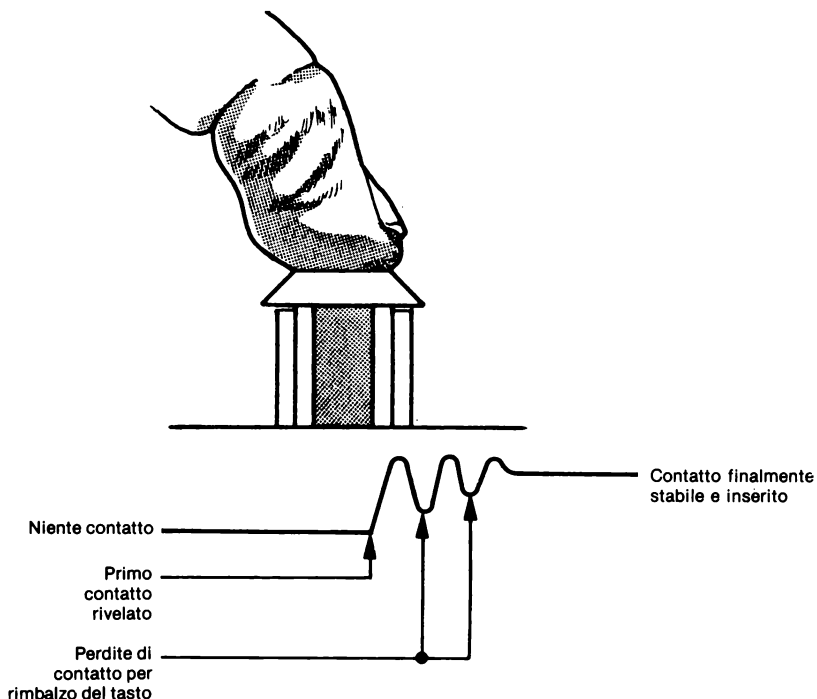
Niente contatto

Esecuzione contatto

Disgraziatamente, i contatti elettrici non sono sempre perfetti. Il contatto rappresentato qui sopra in modo semplicistico potrebbe più verosimilmente verificarsi come segue.

Contatto finalmente stabile e inserito

Niente contatto

Primo contatto rivelato

Perdite di contatto per rimbalzo del tasto

DEBOUNCING

La pressione variabile di un dito può tradursi in un contatto del tasto che appare sobbalzare su e giù per qualche istante dopo che il tasto è stato premuto; questo comportamento è detto rimbalzo. Ogni tastiera degna di questo nome non potrà che contenere l'elettronica per eliminare il rimbalzo (detta "debouncing").

Quindi una tastiera a rimbalzo eliminato traduce un ondeggiante segnale di commutazione in un perfetto scalino off-on, come qui mostrato.

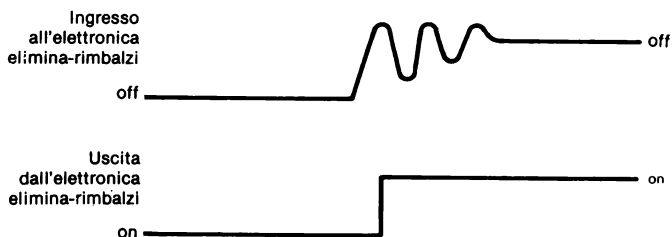
Ingresso all'elettronica elimina-rimbalzi

off

off

Uscita dall'elettronica elimina-rimbalzi

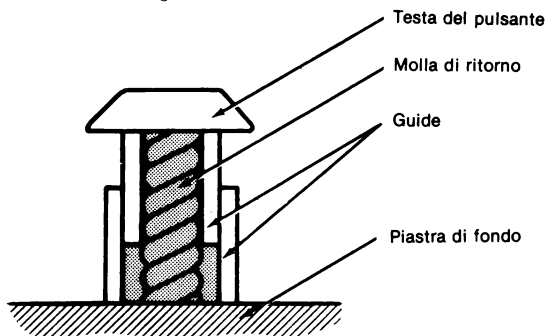
on

on

Il rollover, il buffer e il debouncing sono opzioni elettroniche associate alle tastiere. Ma è ugualmente importante valutare, delle tastiere, gli aspetti e le soluzioni meccaniche. Il tipo più normale di tastiera meccanica ha i tasti che possono essere concettualmente illustrati come segue

Testa del pulsante

Molla di ritorno

Guide

Piastra di fondo

Se entrassimo in dettagli (del resto, non necessari) potremmo scrivere un intero libro semplicemente prendendo come soggetto commutatori meccanici ben progettati e realizzati. Ma al nostro attuale livello di discussione, è necessaria solamente la più superficiale descrizione, ed è proprio a ciò che provvede la figura.

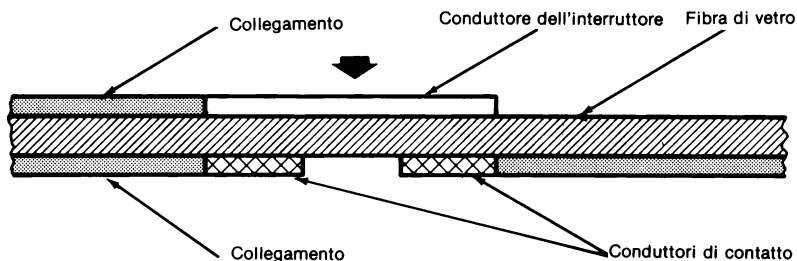
Un commutatore meccanico avrà innanzitutto un qualche tipo di molla che lo riporta nella posizione "disinserito".

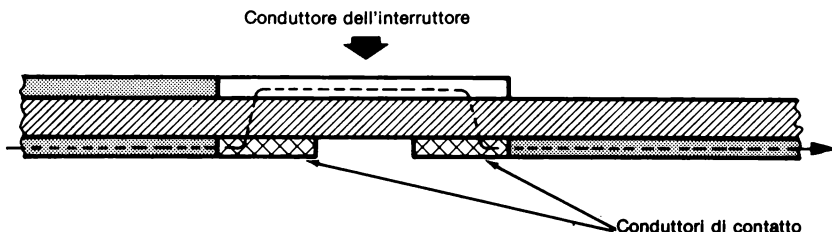
Spingendo in giù il pulsante, la molla verrà compressa; in un qualche punto, il pulsante sarà sufficientemente abbassato da ottenere, in corrispondenza un contatto elettrico: in questo punto, il commutatore è inserito. Dovrà anche essere prevista una guida meccanica sufficientemente robusta per assicurare che, quando viene premuto il pulsante, essa permetta uno scorrimento facile fino al punto in cui si verifica il contatto.
Un commutatore meccanico può anche sembrare un dispositivo molto elementare, ma in realtà non lo è affatto.
Per esempio, nessuno mai premerà un pulsante in modo perfettamente diritto; invariabilmente l'angolo col quale le dita toccano il pulsante farà sì che lo stesso venga spinto verso un lato, oltre che in basso.
Quindi il pulsante va progettato tenendo presente questo. Consideriamo anche il contatto elettrico; potrebbe sembrare che esso si possa realizzare molto semplicemente con del metallo nello stelo del tasto che vada a far contatto con la guida o con la base del fissaggio. Ma questo tipo di contatto a secco, se non è risolto più che bene, non vi darà altro che noie; infatti è sufficiente una modesta corrosione sulle superfici di contatto dei metalli per rendere inefficiente il pulsante.
Talvolta si possono recuperare tastiere economiche spruzzandole con un adatto pulitore di contatti; ma questo non va fatto indiscriminatamente, in quanto può verificarsi che lo spray vada ad attaccare la plastica con cui è costruito il pulsante.
Anche i tasti meccanici presentano opzioni; la più comune consiste in un click udibile o tattile che accompagna l'azionamento di un tasto. Non è cioè il microcomputer che ha bisogno del click, bensì è l'operatore umano; il click c'è semplicemente per assicurare l'operatore che il tasto è stato realmente premuto.
Un tasto può far difetto se la molla di ritorno si indebolisce o si spezza, o se il contatto si sporca.

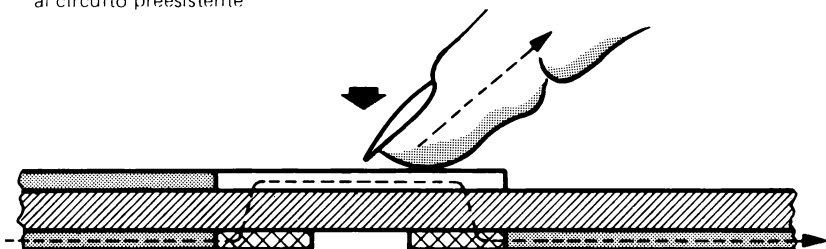
Alcune tastiere sono costruite come unità integrali, tali che anche un solo tasto difettoso obbliga a sostituire l'intera tastiera.
Queste tastiere integrali sono più economiche come primo acquisto, ma ovviamente più costose nella sostituzione se qualcosa non va.

PULSANTI A TOCCO

In un futuro molto prossimo, le tastiere meccaniche saranno molto probabilmente sostituite da pulsanti a tocco. Questo tipo di pulsante consiste in piastrine di fibra di vetro o altro materiale inerte sopra le quali sono designate e stampate delle sagome usando vernici elettricamente conduttrici, secondo quanto qui sotto illustrato.

Collegamento
Conduttore dell'interruttore
Fibra di vetro
Collegamento
Conduttori di contatto

Normalmente una corrente elettrica entra da uno dei contatti, passa attraverso l'isolante e va al conduttore superiore, e di qui, ripassando attraverso la piastrina isolante, torna all'altro contatto ed esce dal secondo connettore.

Conduttore dell'interruttore
Conduttori di contatto

La corrente d'uscita è fortemente distorta dal vetro, ma le sue caratteristiche sono perfettamente riconoscibili ad una apposita logica elettronica.

Appoggiando un dito sul commutatore, il vostro corpo va a costituire un'aggiunta al circuito preesistente.

Il tocco del dito sul pulsante altera drasticamente la corrente d'uscita, la logica elettronica esterna rivela questo cambiamento di corrente d'uscita e lo traduce nella situazione di commutatore aperto, o inserito.

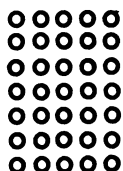
OPZIONI PER LA STAMPANTE

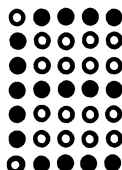
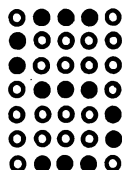
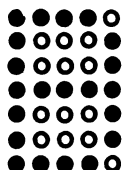
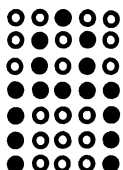
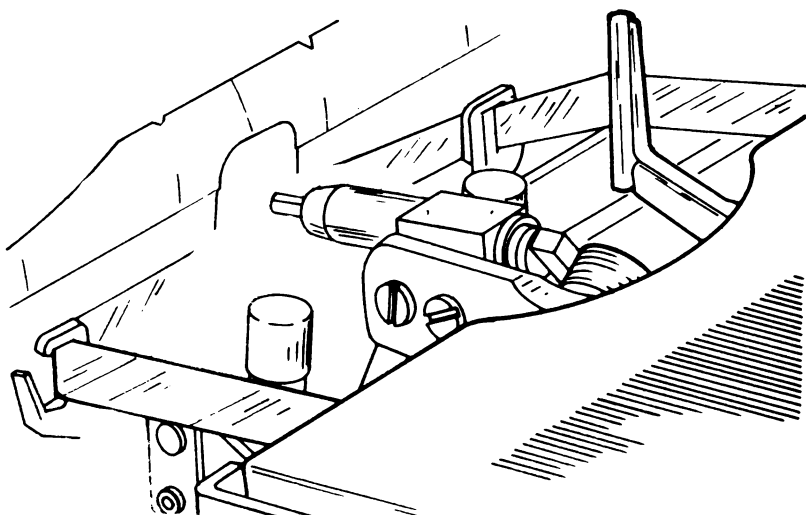
Esistono tre alternative di stampante che influiscono in modo più significativo sul prezzo: il meccanismo di stampa, la linea (e la carta) e la velocità di stampa.

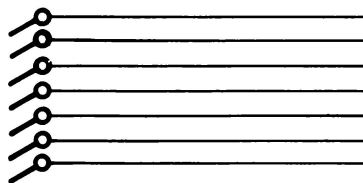
MECCANISMI DI STAMPA

Cominciamo dai meccanismi di stampa. Ci sono numerosi modi in cui le stampanti possono creare i caratteri sulla carta; noi esamineremo solo i meccanismi di stampa comunemente adottati nelle stampanti a basso costo.

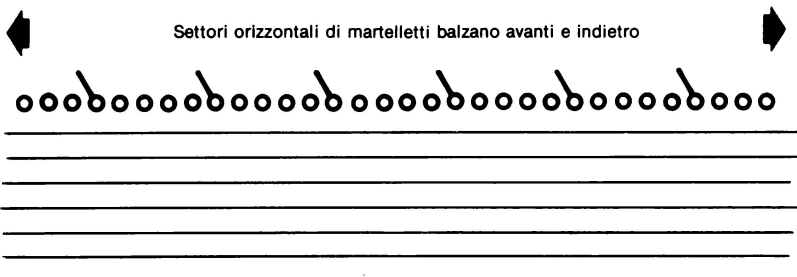
STAMPANTI A MATRICE

La maggior parte delle stampanti di basso costo producono caratteri come matrici di punti.
Le più semplici stampanti a matrice creano caratteri da una matrice di 5 per 7 punti.

Ciascuna posizione della matrice è un punto nel quale la stampante può stampare un singolo contrassegno

Qui sono rappresentati caratteri che possono essere generati da una matrice a 5x7 punti.

La matrice a 5x7 punti è adeguata per lettere maiuscole, numeri e caratteri speciali, ma produce lettere minuscole mal definite. Le matrici a 7x9 punti forniscono naturalmente una miglior definizione.
Le stampanti a matrice di punti possono essere del tipo a battuta o non a battuta. Una stampante a battuta, come implica il suo nome, produce segni sulla carta percuotendo la carta attraverso un nastro inchiostrato; per tale scopo è usato un piccolissimo martelletto.

Per creare i caratteri delle matrici a punti sono comunemente usate due tecniche; una usa una linea verticale di martelletti che "spazzola" la carta, linea per linea, creando i caratteri.

Questi sono sette martelletti di stampa

7 martelli di stampa viaggiano sulla carta creando una singola linea di carattere

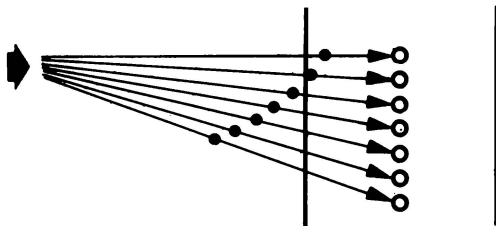
L'altro tipo di stampante ha un fascio orizzontale di martelletti che saltellano avanti e indietro creando i caratteri come segue:

Settori orizzontali di martelletti balzano avanti e indietro

Il vantaggio delle stampanti a percussione consiste nella possibilità di eseguire copie multiple di qualunque cosa si stampi, usando carta carbonata o carta sensibile alla pressione, esattamente come per una macchina da scrivere.
Lo svantaggio è che esse sono relativamente lente, poiché la stampa coinvolge martelletti che si devono muovere. Una normale stampante a matrici può stampare indifferentemente fra 30 e 300 caratteri al secondo.
Esistono anche tipi di stampanti non a battuta che creano segni sulla carta usando una varietà di tecniche più o meno ingegnose, nessuna delle quali comunque prevede la percussione della carta con qualsiasi oggetto.
Queste stampanti a matrice non a percussione sono molto veloci in quanto, per stampare i punti, esse non devono far muovere alcuna parte meccanica; ma non possono essere usate per stampare copie multiple in quanto non c'è nulla che colpisca la carta abbastanza forte per effettuare copie.

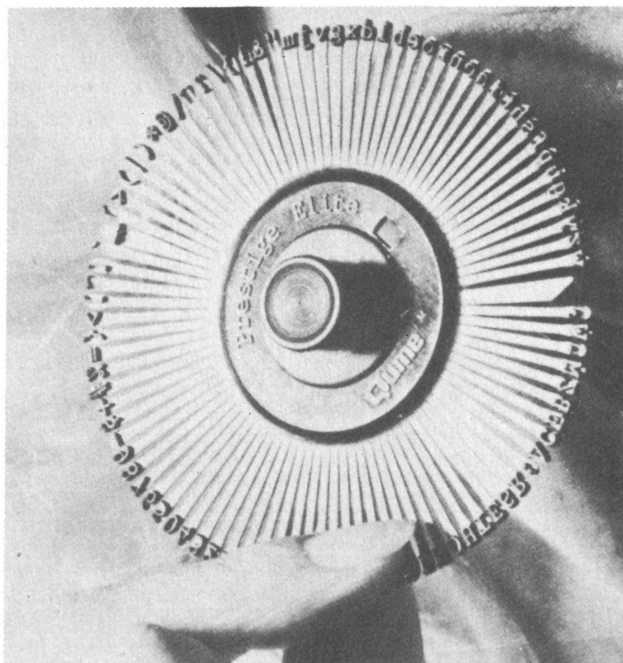
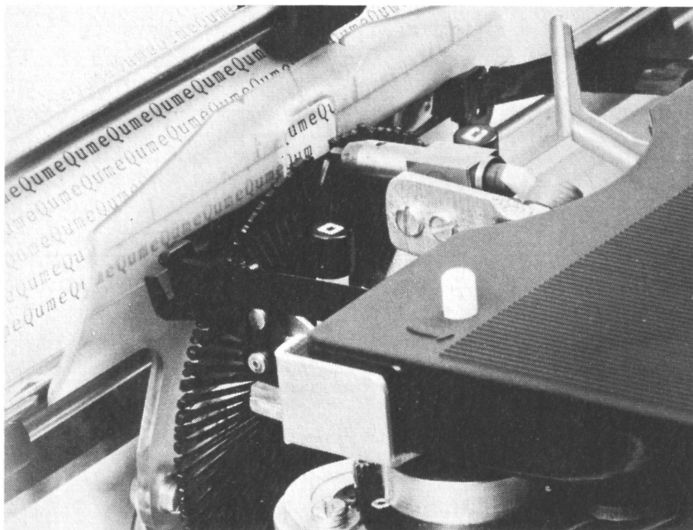
STAMPANTI TERMICHE

Alcuni tipi di stampanti a matrici non a percussione, molto economiche, usano speciali carte termosensibili. Queste stampanti hanno una testina che si muove attraverso la carta emanando impulsi di calore ovunque debba essere creato un punto. Le stampanti che usano questa tecnica sono chiamate termiche.

STAMPANTI A GETTO DI INCHIOSTRO

Le stampanti a spruzzo di inchiostro sono le più comuni stampanti non a battuta disponibili oggi. Che lo crediate o no, una stampante a getto d'inchiostro stampa sparando un fiotto ad alta velocità di minuscole goccioline di inchiostro sulla carta. Ciascuna goccia d'inchiostro, che porta una piccola carica elettrica, passa attraverso un campo magnetico che le fa deflettere così che colpiscano la carta esattamente nella posizione voluta, come indica l'illustrazione.

Le gocce sparate dalla stampante a getto d'inchiostro sono minuscole e seccano quasi immediatamente; queste goccioline vengono sparate così velocemente che questo tipo di stampante può stampare migliaia di caratteri al secondo.

STAMPANTE A RUOTA MARGHERITA (DAISY WHEEL)

Alcuni costruttori statunitensi producono un tipo di stampante detta a "ruota margherita". Essa usa un elemento stampante che assomiglia ad una margherita a 96 petali; all'estremo di ogni petalo c'è un carattere.

Le stampanti a ruota margherita fanno girare l'elemento stampante ad alta velocità mentre lo fanno spostare sul foglio di carta. Non appena il petalo esatto è di fronte alla posizione del carattere, viene azionato un martelletto che fa battere il petalo contro un nastro che stampa il carattere sul foglio.

I vantaggi delle stampanti a margherita consistono nel fatto che esse producono stampati molto eleganti, sono molto affidabili e relativamente economiche.

I testi stampati da una ruota a margherita risultano chiari ed attraenti, cosa che non può dirsi per le stampanti a matrice: **la ragione è nel fatto che le stampanti a ruota generano caratteri completi.**

Le stampanti a ruota possono lavorare per migliaia di ore senza guasti meccanici, mentre le macchine da scrivere modificate per operare come stampanti possono presentare ripetutamente dei difetti.

L'unico svantaggio delle stampanti a margherita è che sono molto lente; le velocità tipiche di stampa sono infatti sui 30 ÷ 45 caratteri al secondo.

STAMPANTI A NASTRO D'ACCIAIO

Un altro modo, più costoso, di generare caratteri interi quali quelli della stampante a margherita è quello di usare un nastro d'acciaio. Una stampante a nastro fa ruotare un nastro d'acciaio ad alta velocità, con uno o più martelletti che colpiscono il nastro ogniqualvolta il carattere appropriato si trovi esattamente davanti alla sua posizione corretta.

OPZIONI COMUNI

Tutti i tipi di stampanti a basso costo oggi disponibili usano una delle tecniche di stampa qui descritte: ma qualunque sia il meccanismo di stampa, esistono alcune opzioni più comuni che dovete conoscere prima di scegliere una stampante.

LUNGHEZZA DELLA LINEA STAMPATA

1) Lunghezza della linea.
Linee di 80 caratteri sono le più comuni; alcune stampanti generano linee di 132 caratteri. Certe stampanti di basso costo scrivono linee di 20 caratteri, usando nastri piuttosto stretti.

TIPO DI CARATTERE

2) Solo maiuscolo, o maiuscolo e minuscolo.
Le stampanti economiche possono stampare solo le lettere maiuscole, mentre ne esistono di più costose che possono stampare sia caratteri maiuscoli che minuscoli.

STAMPANTI A DUE TESTINE

3) Numero delle testine di stampa.
Le stampanti a matrice ed a ruota, che viaggiano attraverso il foglio orizzontalmente, normalmente presentano versioni a doppia velocità ed a costo più elevato. Le versioni a doppia velocità usano due testine stampanti, ognuna delle quali esplora metà della linea da stampare.

STAMPA A LINEE INVERTITE

La maggior parte delle stampanti con le testine che spazzolano il foglio di carta orizzontalmente aumentano la loro velocità di stampa viaggiando avanti e indietro attraverso il foglio.

Al contrario una normale macchina da scrivere stampa sempre da sinistra a destra, effettuando un ritorno carrello alla fine di ogni linea.

SPAZIATURA PROPORZIONALE

4) Spaziatura proporzionale: attualmente questa possibilità è disponibile solo sulle stampanti a ruota margherita. Il testo proporzionalmente spaziato lascia per ciascun carattere una spaziatura differente, che dipende dalla forma e dimensione di tale carattere.

Riproduciamo qui un esempio di stampato eseguito con stampante a spaziatura proporzionale.

(2) The ending location is an address

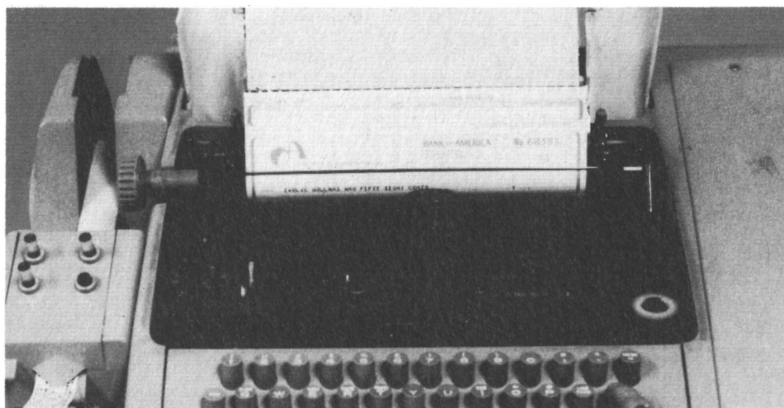
(2) The ending location is an address

5) Regolazione bordo destro.
Il testo di quest'articolo, come della maggioranza di libri e riviste, è composto in modo che sia il margine di sinistra che quello di destra delle colonne di stampa costituiscano delle linee diritte e parallele.

Per le stampanti di basso costo invece, se ci si può sempre aspettare che ciascuna riga parta (sulla sinistra) nello stesso punto di allineamento, non si verificherà, in genere, che tali righe terminino tutte allo stesso punto.

La regolazione del bordo destro è il termine che si applica a quei testi che presentano le righe che iniziano e terminano su linee diritte e parallele.

AVANZAMENTO CARTA A FRIZIONE

6) Avanzamento del foglio
Se si deve stampare su un foglio con continuità, come nel caso degli assegni di Joe Bitburger, si può facilmente cadere in problemi di posizionamento del foglio stesso. Infatti le stampanti economiche fanno avanzare il foglio usando un movimento a frizione, mediante due piccoli rulli che spingono sulla carta facendola spostare.

Anche una piccola percentuale di slittamento può quindi far capitare il foglio dove non dovrebbe esattamente essere; linee successive vengono quindi stampate in posizioni via via sempre più lontane da quelle desiderate.

Sull'assegno dell'esempio qui sotto riportato, va stampata una riga dove dovrebbe apparire l'ammontare in dollari. Se ciascun assegno slitta anche solo di 1 millimetro (il che è tutto dire) esso avanzerà di 1 millimetro in meno per ogni riga; quindi dopo 5 assegni, l'ammontare in dollari apparirà scritto circa mezzo centimetro più in alto della sua posizione corretta. Dopo un po' di tempo, l'assegno diventerà illeggibile.

NO. 382

MARCH 8 19 77

AMOUNT $ 103. 75

DOLLARS

NO. 382

MARCH 8 19 77

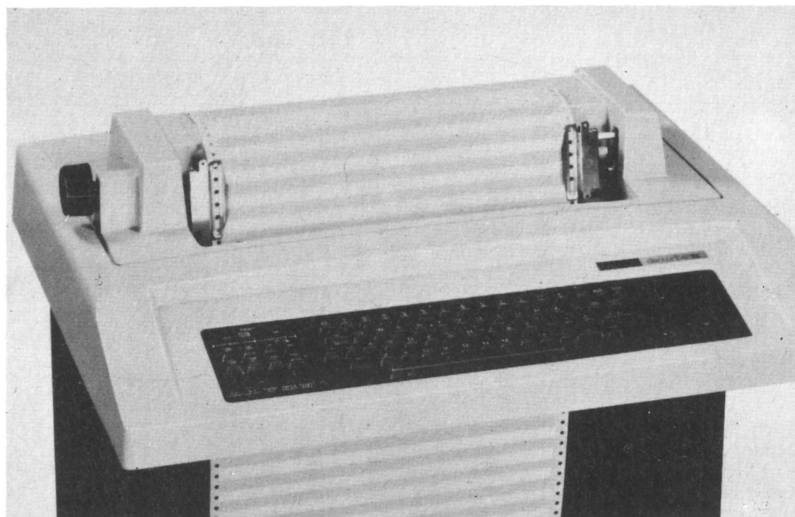
103. 75

AMOUNT $

DOLLARS

AVANZAMENTO CARTA A RUOTA DENTATA

In questi casi, di necessità di stampare con continuità, è consigliabile disporre di avanzamenti a ruote con perni sporgenti.

In tal caso, poiché i pernetti entrano esattamente nella perforazione sui bordi del foglio, il trascinamento è esatto e si ripete identicamente all'infinito.

7) Formato pagina.

Alcune stampanti incorporano dei sistemi meccanici per specificare il formato della pagina di stampa, cioè le dimensioni occupate, sul foglio, della zona stampata.

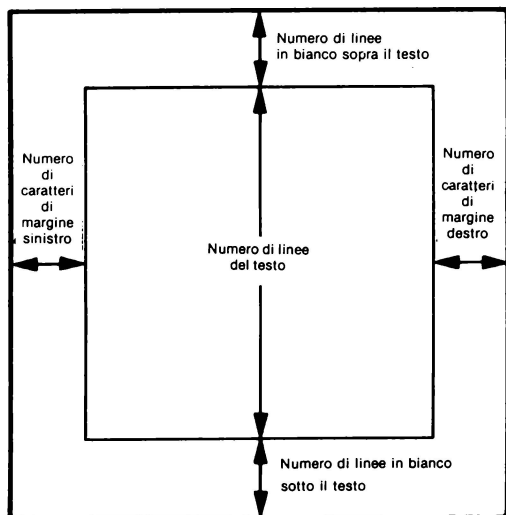
Il formato pagina include possibilità come: il numero di linee su ciascun foglio di carta, l'ampiezza dei margini superiore ed inferiore, nonché dei margini sinistro e destro.

Numero di linee in bianco sopra il testo

Numero di caratteri di margine sinistro

Numero di linee del testo

Numero di caratteri di margine destro

Numero di linee in bianco sotto il testo

Dopo la valutazione che abbiamo effettuato delle varie opzioni disponibili nel settore stampanti, rimane ancora da considerare la solidità delle stesse.

Bisogna ricordare che la stampante è un dispositivo meccanico, e sarà sempre la parte meccanica del vostro sistema calcolatore che si guasterà per prima.

Se una stampante non è solidamente costruita, non importa quante favolose prestazioni vi possa offrire: si guasterà presto.

Quindi, prima di acquistare stampanti economiche, occorre assicurarsi che l'economia non sia stata fatta a scapito non tanto delle possibilità quanto della qualità.

Per esempio, date prima un'occhiata alla "quantità" di plastica usata: la plastica è bella e buona per la scatola esterna e qualche particolare generico, ma non per i meccanismi di stampa, che devono essere di metallo.

OPZIONI PER LA MEMORIA DI MASSA

Abbiamo già discusso tre tipi di dispositivi per memoria di massa: unità a nastro perforato, unità a cassette e floppy disk.

Le unità a disco rigido sono raramente usate con i sistemi microcomputer, in quanto esse surclassano il microcomputer sia in termini di velocità di trasferimento dati che in capacità di immagazzinare informazioni.

UNITA' A NASTRO DI CARTA

Gli unici lettori e perforatori di nastri che si vedono un po' ovunque sono unità a basse prestazioni ed a bassissimo costo.

Ciò avviene in quanto le unità a cassette sono così chiaramente superiori che difficilmente qualcuno va a pagare, per un'unità a nastro perforato, lo stesso prezzo di un'unità a cassette.

Di conseguenza, le alternative disponibili per chi desidera usare nastro perforato sono molto poche. Il sistema microcomputer della Heathkit usa anche un perforatore di nastro; è una delle poche alternative disponibili.

Esistono anche alcuni perforatori meccanici economici che permettono di perforare i nastri a mano.

Spostandoci da questi dispositivi piuttosto elementari, si troveranno più facilmente alternative valide; per esempio, un terminale teletype, completo di lettore e perforatore di nastro, è reperibile usato (almeno sul mercato USA) a prezzi anche nettamente inferiori al milione.
Questo prezzo non lascia molto spazio ai costruttori di unità a nastro per produrre qualcosa di economico e valido allo stesso tempo.

E neppure si prevede un gran futuro nel tentar di costruire un'unità a nastro di carta che possa essere acquistata perché perfora e legge molto velocemente: si possono infatti acquistare unità di riproduzione a cassette che sono più veloci ed egualmente economiche.
Come già accennato nel 1° capitolo, si può anche usare un qualunque registratore a cassette del mercato che, assieme ad una appropriata elettronica di interfacciamento, costituisce un buon dispositivo di memoria di massa per qualsiasi sistema microcomputer.

Il riproduttore di cassette, da solo, anche se di buona qualità, costerà meno di 100.000 lire (ci si riferisce sempre al mercato americano); le interfaccie elettroniche sono già scese a prezzi piuttosto bassi. In conclusione il mercato per le unità a nastro di carta si presenta molto limitato.

UNITA' A CASSETTE

C'è invece, come unità di memoria di massa per sistemi a microcomputer, un considerevole mercato per i riproduttori a cassette. Al momento attuale, le unità a floppy disk più economiche costano, assieme all'elettronica d'interfaccia, oltre un milione.

Si prevede, per i floppy, una sensibile diminuzione nei costi, visto l'aumento nel volume di vendite; ma nel frattempo (e ancora per molto) le unità a cassette domineranno come memorie di massa a basso costo.

Quando si deve scegliere un'unità a cassette per un sistema microcomputer, ci sono due importanti considerazioni da fare.

1) **Occorre assicurarsi che il giranastri sia di buona qualità.** Non conviene usare tipi troppo economici, che potrebbero dare nient'altro che seri grattacapi.

2) **Occorre accertarsi che la logica di interfaccia per il vostro riproduttore memorizzi informazioni in record piuttosto brevi, usando la registrazione ridondante;** nel 1° capitolo abbiamo discusso i vantaggi di questa procedura. Bisogna anche essere molto cauti nel caso di unità a cassette che offrano velocità molto elevate di trasferimento dati; come norma, l'affidabilità aumenta come diminuisce la velocità di trasferimento, quindi un riproduttore lento è un riproduttore affidabile.

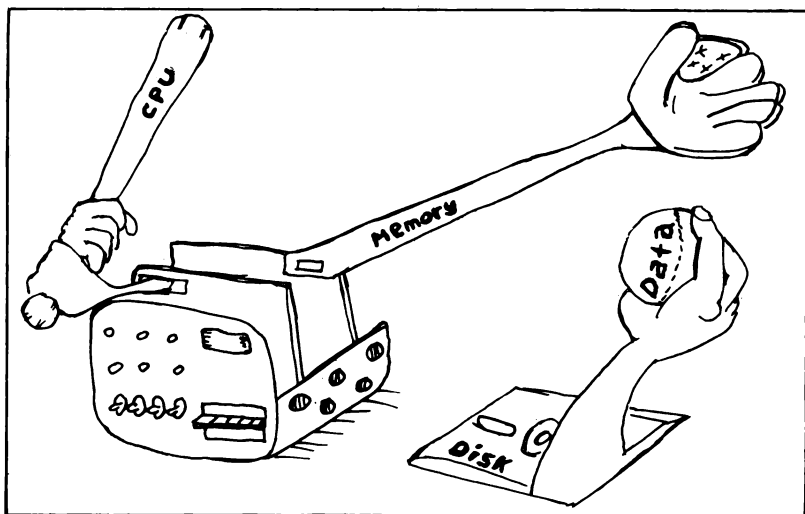
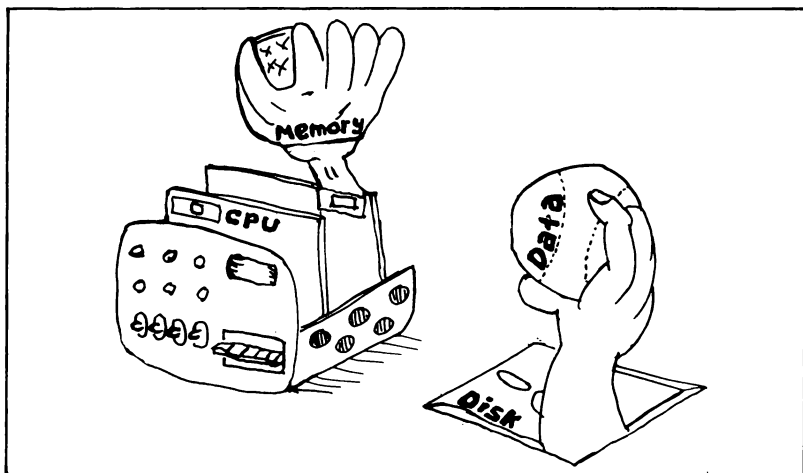
COMPATIBILITA' DEI DISPOSITIVI

Quando si compra un'unità a cassette per un sistema microcomputer, è opportuno controllare quanti altri modelli di unità a cassette sono in grado di leggere le cassette prodotte dal vostro e di registrare le cassette che il vostro può leggere. Questo è ciò che s'intende per compatibilità dei dispositivi.

Acquistando un'unità a cassette per il vostro sistema microcomputer, in genere è possibile ottenere un elenco di tutti i modelli di registratori compatibili col vostro; e la compatibilità è un'utile caratteristica perchè significa che voi potete scambiare cassette con un più ampio numero di altri utenti.

UNITA' A FLOPPY DISK

L'alternativa unità a disco floppy è già stata in gran parte esaminata nel primo capitolo. In questo caso, occorre innanzitutto decidere se ci serve una versione minifloppy o una versione floppy disk standard; presa questa decisione, anche in questo caso vale la pena di determinare quante unità floppy disk sono compatibili con quella acquistata da voi.

Un'opzione importante da esaminare all'atto dell'acquisto di un'unità floppy disk e il modo in cui la logica d'interfaccia trasferisce l'informazione fra il disco e la memoria. Alcuni controllori di unità a floppy disk effettuano tutto il trasferimento dati attraverso il microcomputer.

Questo però è un sistema lento e sconsigliabile di smistare dati. Unità floppy disk migliori sono quelle capaci di smistare l'informazione direttamente fra la memoria leggi/scrivi ed il disco floppy, saltando così il microcomputer.

Questo è il sistema preferenziale di lavoro per un controllore di floppy disk. Accedendo alla memoria direttamente, la logica d'interfaccia del floppy può smistare dati più velocemente, lasciando nel contempo il microcomputer libero di eseguire programmi mentre è in corso il trasferimento dati.

E' anche opportuno esaminare il modo in cui è costruito il sistema che pilota il floppy disk.

Bisogna infatti ricordare che si tratta di un dispositivo meccanico, e oltretutto anche di un'unità ad alta precisione che deve essere capace di muovere una testina di lettura molto rapidamente sulla superficie del disco rotante. Poiché la precisione deve essere dell'ordine dei centesimi di millimetro, se i meccanismi di rotazione e di controllo fossero costruiti con plastica economica, essi si consumerebbero rapidamente, con conseguenze disastrose.

Poiché le unità a floppy disk sono piuttosto costose, vi è un motivo in più per verificare che gli elementi di controllo meccanico siano stati costruiti usando componenti metallici di alta qualità; ancora una volta (e a maggior ragione) la plastica è OK per il mobiletto e poco altro.

Capitolo 4
GETTANDO LE BASI

Si vuole ora guardare all'interno di un microcomputer ed osservare come lavora. Queste informazioni possono non essere richieste.

Si può acquistare un microcomputer ed usarlo come tale, scrivendo programmi in un linguaggio di programmazione (come il FORTRAN oppure il BASIC); se questo è l'obiettivo da raggiungere si può non conoscere come il sistema lavora — ed il resto di questo libro avrà meno importanza in questo caso. Dopo aver letto le prime pagine del capitolo 5, inerenti ai linguaggi di programmazione, basterebbe reperire un libro sul linguaggio di programmazione su cui si vuol basare il lavoro.

In realtà la programmazione di un sistema microcalcolatore mediante un linguaggio come il FORTRAN o il BASIC è come viaggiare su un autobus. Per viaggiare su un autobus non si deve conoscere come esso lavora; si può anche non conoscere come guidarlo. Comunque, se si vuole aumentare la flessibilità di un'auto privata, si deve imparare a guidarla; si deve anche imparare come la macchina lavora in modo da realizzare l'adattamento ottimo. Analogamente se si vuole accedere alle caratteristiche ed alla potenza del proprio microcomputer — piuttosto che accedere a potenze e caratteristiche più limitate di un linguaggio di programmazione — è strettamente necessario imparare come lavora il microcomputer.

Se si decide di apprendere come lavora il microcomputer, si deve identificare il livello di conoscenza ricercato. Si può imparare come programmare il microcomputer impiegando un linguaggio di programmazione a livello di microcomputer: si può fare un passo ulteriore ed imparare come il microcomputer lavora in modo sufficientemente dettagliato da valutare i limiti — e di espanderlo o cambiarlo se esso non soddisfa le richieste.

La parte successiva di questo libro presuppone che si voglia apprendere come operano i microcomputer — ma non che si voglia adattare o modificare il microcomputer. Si parte inoltre dal presupposto che i lettori di questo libro non abbiano alcuna base sui microcomputer ma che abbiano assimilato i capitoli 1, 2 e 3. Su queste basi si spiegheranno alcuni concetti veramente fondamentali. Queste spiegazioni mirano al superamento del gap per la lettura e la comprensione di "Un'introduzione ai microcomputers: Volume 1 — Concetti di base". Così, presumendo che si voglia fare molto di più che programmare un microcomputer per mezzo di un linguaggio ad alto livello, è necessaria la lettura del seguito di questo libro, indipendentemente dagli obiettivi specifici. Solo dopo essere arrivati ai libri successivi della serie, inizierà la discriminazione tra ciò che si vuole e ciò che non si vuole fare, in dipendenza delle proprie aspirazioni.

Durante la descrizione dei concetti fondamentali, nel seguito del libro, si farà riferimento continuo al microcomputer stesso, ma non alle unità fisiche residenti nell'interno di esso. Questo è corretto in quanto si vuole imparare a programmare un microcalcolatore ed eventualmente a costruirne uno. Sarà necessario molto più tempo per avere informazioni sui floppy disk, tastiere, displays ed altre unità fisiche oppure la loro logica di interfaccia. Si comprerà l'unità fisica e la sua logica di interfaccia come un'unità singola e si programmerà l'unità fisica del microcalcolatore.

I NUMERI E LA LOGICA

Ciascun circuito logico all'interno di un sistema microcomputer si può ridurre ad un insieme di interruttori, ciascuno dei quali è acceso o spento.

Questo concetto non vi giungerà del tutto nuovo, in quanto lo abbiamo già usato per descrivere le memorie e, specificatamente, le R.O.M.

Ma esaminiamo ora come un circuito di interruttori può in concreto realizzare la potenza e la versatilità di un computer.

DATI BINARI

Consideriamo i numeri; come abbiamo spesso stabilito, le istruzioni, i programmi e i dati di qualsiasi tipo diventano una sequenza di numeri.

Come faremo a rappresentare tanti numeri visto che non possediamo altro che interruttori che possono essere accesi o spenti?

Il digit "0" può essere rappresentato da un interruttore spento:

il digit 1 può rappresentare un interruttore acceso:

E allora? Un computer che può contare solo fino a 1 non potrà essere molto utile. In realtà, i computer possono dar luogo solo a due digit numerici separati e distinti:

zero	0
uno	1

Ma gli uomini hanno un problema molto simile; essi sono limitati a non più di dieci digit numerici distinti e separati:

zero	0
uno	1
due	2
tre	3
quattro	4
cinque	5
sei	6
sette	7
otto	8
nove	9

ed ora iniziano i digit che si combinano

dieci	10
undici	11
ecc.	

NUMERI DECIMALI

Questo sistema usato dagli uomini è definito sistema numerico decimale. Esso compare in tutto il mondo, fra la totalità delle nazioni e delle tribù, ovunque le società abbiano imparato a contare.

Molto probabilmente ciò è dovuto al fatto di avere tutti 10 dita e di aver imparato a contare su queste. Non c'è nulla di "naturale" o di "unico" nel sistema numerico decimale; in effetti, è un modo piuttosto grossolano di fare le cose: infatti vedremo più avanti che esistono sistemi di calcolo molto più appropriati, nel senso di rendere l'aritmetica più facile.

Così come ha un nome il sistema di calcolo umano — sistema decimale — così ha un nome anche il **sistema di conteggio dei computer, ed è sistema binario. Guardiamo in dettaglio tale sistema.**

IL SISTEMA NUMERICO BINARIO

Il sistema binario ha solo due digit distinti e separati: 0 e 1; per rappresentare i numeri più grandi di 1, non si fa che seguire l'esempio dei numeri decimali, ed usare più di un digit. Consideriamo il numero due; in forma binaria esso è rappresentato dai digit 10:

DECIMALE		BINARIO	
zero	0	zero	0
uno	1	uno	1
due	2	due	10
tre	3		
quattro	4		
cinque	5		
sei	6		
sette	7		
otto	8		
nove	9		
dieci	10		

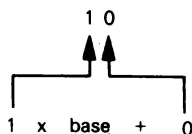
Sia nel sistema di conteggio decimale che in quello binario, la combinazione di due digit "10" rappresenta un numero che è superiore di uno al numero più alto rappresentato da un solo digit. Nel caso dei numeri decimali il numero più alto a singolo digit è nove; perciò 10 rappresenta uno più di nove. Nel sistema binario il più alto numero a singolo digit è uno; quindi 10 rappresenta uno più di uno, e cioè due.

Il valore numerico della combinazione "digitale" 10 è molto importante. Tale condizione di digit ha valore 10 nel sistema decimale, che è quello dal quale tale sistema prende il suo nome; similmente, nel sistema binario, la combinazione "10" ha il valore di due, da cui proviene il nome del sistema binario.

NUMERO BASE

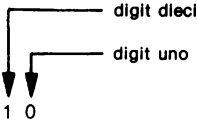
Questi valori, dieci per il sistema decimale e due per il sistema binario, sono indicati come "base" per il sistema numerico.

Il numero base, cioè il valore associato con la combinazione di digit "10", viene interpretata per i numeri decimali o binari come segue.

```
          1 0
          ↑ ↑
   ┌──────┘ └──────┐
   │               │
   1  x  base  +   0
```

DIGIT UNO
DIGIT DIECI

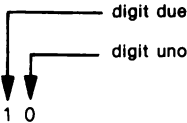
Ci si riferisce ai digit di un numero decimale a due digit come digit "uno" e digit "dieci".

digit dieci
digit uno

1 0

DIGIT DUE

Per un numero binario a due digit, ci si riferisce ai digit come digit "uno" e digit "due".

digit due
digit uno

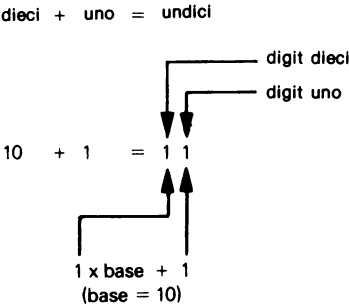
1 0

Per rappresentare tre, usando numeri binari, possiamo ancora tracciare un parallelo col nostro sistema di calcolo decimale.

Il numero decimale successivo a 10 è creato aggiungendo 1, come segue:

dieci + uno = undici

digit dieci
digit uno

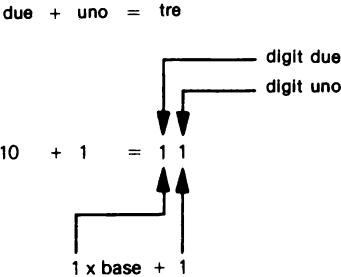
10 + 1 = 1 1

1 x base + 1
(base = 10)

Similmente, nell'uso dei numeri binari, noi avanziamo da due a tre aggiungendo 1 al digit binario due:

due + uno = tre

digit due
digit uno

10 + 1 = 1 1

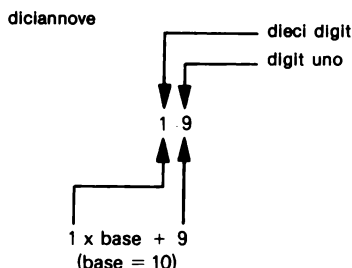
1 x base + 1

Cosa succede quando vogliamo creare 4 usando numeri binari? In questo caso, il parallelò col sistema decimale non è più così immediato. Infatti, proseguendo col decimale11, abbiamo ancora molta strada da percorrere prima che il problema

sorga; e possiamo andare avanti ad aggiungere 1 finché non raggiungiamo il decimale diciannove.

diciannove

dieci digit
digit uno

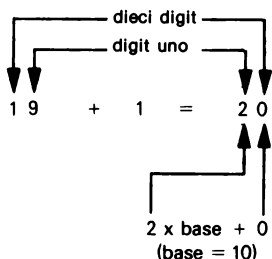
1 9

1 x base + 9
(base = 10)

Poi passiamo al decimale venti.

diciannove + Uno = venti

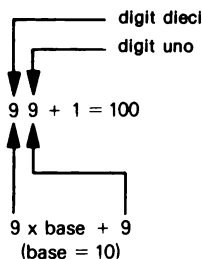
dieci digit
digit uno

1 9 + 1 = 2 0

2 x base + 0
(base = 10)

E' solo quando si è raggiunto il decimale 99 che si deve aggiungere un terzo digit e creare così 100:

digit dieci
digit uno

9 9 + 1 = 100

9 x base + 9
(base = 10)

Ora, nel sistema binario, procedere dopo il binario 11 (che è decimale 3) porta ad un problema; se aggiungiamo 1 a 11, non possiamo ottenere 12, poiché il digit 2 non esiste nel sistema binario. E inoltre, non possiamo passare da 11 a 20, perché ancora una volta useremmo il 2, che è fuori legge come digit binario.

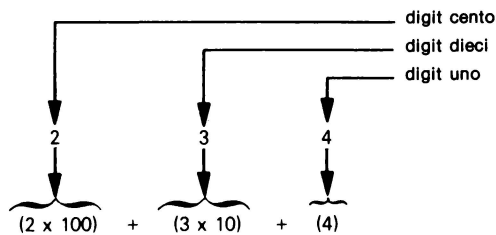
I numeri binari devono perciò far seguire all'**11 il 100:**

zero	0
uno	1
due	10
tre	11
quattro	100

Esaminiamo il significato dei numeri a tre digit. Quando vedete il numero 234, automaticamente lo interpretate come duecentotrentaquattro.

digit cento
digit dieci
digit uno

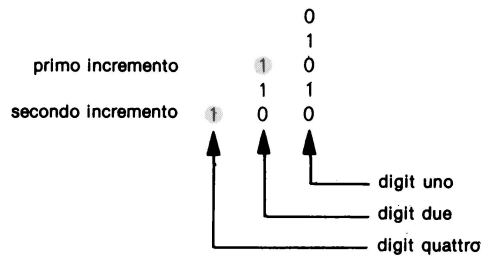
2 3 4

(2 x 100) + (3 x 10) + (4)

Ma c'è un significato speciale nel "digit "centinaia", esattamente come c'è nel digit "decine". Si possono incrementare i digit delle decine nove volte; al decimo incremento bisogna incrementare anche il digit delle centinaia. Consideriamo gli incrementi dei digit delle decine, cominciando col numero decimale tre:

			3	tre
primo incremento		1	3	tredici
secondo incremento		2	3	ventitre
terzo incremento		3	3	trentatre
quarto incremento		4	3	quarantatre
quinto incremento		5	3	cinquantatre
sesto incremento		6	3	sessantatre
settimo incremento		7	3	settantatre
ottavo incremento		8	3	ottantatre
nono incremento		9	3	novantatre
decimo incremento	1	0	3	centotre

Nel caso di numeri binari, potete incrementare i digit "due" solo una volta; col secondo incremento, dovete creare un digit "quattro".

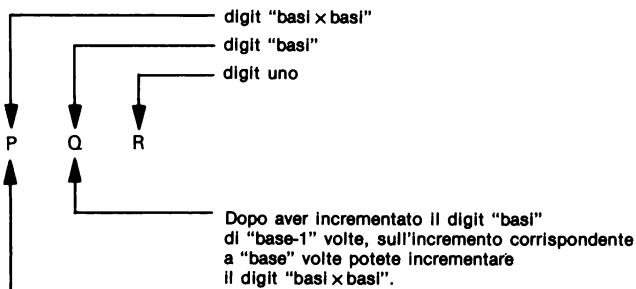
			0
			1
primo incremento		1	0
		1	1
secondo incremento	1	0	0

digit uno
digit due
digit quattro

Questa è la regola: il numero di volte che potete incrementare un digit è uguale a uno meno del numero base; poi, dovete incrementare il digiti successivo più alto.

Cosicché, in un numero decimale potete incrementare ciascun digit nove volte (da zero a 9); poi dovete incrementare il più alto digit decimale successivo. Talché i digit si possono rappresentare come segue:

digit "basi x basi"
digit "basi"
digit uno

P Q R

Dopo aver incrementato il digit "basi" di "base-1" volte, sull'incremento corrispondente a "base" volte potete incrementare il digit "basi x basi".

In questa illustrazione, PQ ed R rappresentano i digit di qualsiasi sistema numerico, sostituendo 2 o 10 alla "base", l'illustrazione rappresenterà numeri binari o decimali. Il secondo digit di un multidigit diventa un moltiplicatore di "numero base" entro un'equazione che vi dice il valore dei numeri multidigit.

Similmente un terzo digit diventa un moltiplicatore per il numero base moltiplicato per se stesso; ciò può illustrarsi come segue:

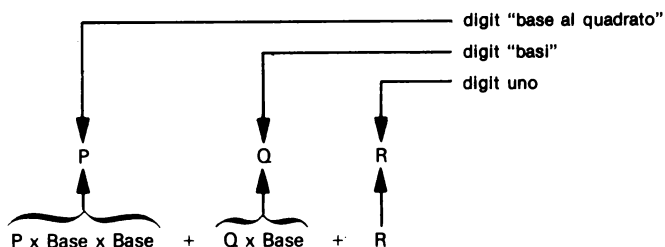
Interpretare il numero binario a tre digit come segue:	1 x (2x2)	+	0x2	+	1	"numero base" =2 (due)
Qui c'è un numero binario a tre digit	1		0		1	
Questo è ciascun numero a tre digit	P		Q		R	
Qui c'è un numero decimale a tre digit	2		3		4	
Interpretare il numero decimale a tre digit come segue	2 x (10x10)	+	3 x 10	+	4	"numero base" =10 (dieci)

QUADRATO DI UN NUMERO

Un numero moltiplicato per se stesso è indicato come il "quadrato" del numero; cosicché **possiamo rappresentare qualsiasi numero a tre digit con la seguente equazione di uso generale:**

digit "base al quadrato"
digit "basi"
digit uno

P Q R

P x Base x Base + Q x Base + R

Per "base x base" usiamo il simbolo $base^2$, cosicché $base^2$ rappresenta il quadrato del numero rappresentato da "base"

DIGIT MIGLIAIA

CUBO DI UN NUMERO

Possiamo estendere lo stesso ragionamento a numeri più grandi. Nell'aritmetica decimale un quarto digit identifica le migliaia, e lo identifichiamo quindi come il digit "migliaia" Per esempio, 2345 rappresenta due migliaia, tre centinaia e quarantacinque.

Un migliaio è 10 x 10 x 10, che è lo stesso che dire "numero base"x"numero base"x"numero base", e cioè il "cubo" di un numero base.

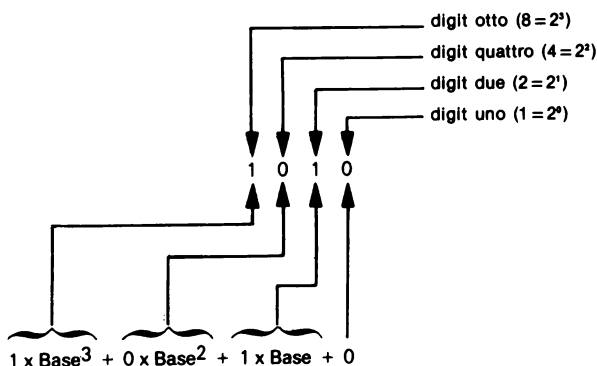
Per "base x base x base" usiamo il simbolo $base^3$; così $base^3$ rappresenta il cubo del numero rappresentato da "base".

DIGIT DEGLI OTTO

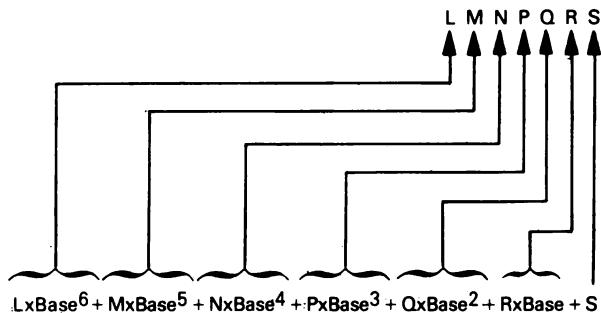
Un numero binario a quattro digit sarà cioè interpretato come segue:

digit otto ($8=2^3$)
digit quattro ($4=2^2$)
digit due ($2=2^1$)
digit uno ($1=2^0$)

1 0 1 0

$1 \times Base^3 + 0 \times Base^2 + 1 \times Base + 0$

Ora possiamo definire i numeri decimali e binari multidigit come segue:

L M N P Q R S

$L \times Base^6 + M \times Base^5 + N \times Base^4 + P \times Base^3 + Q \times Base^2 + R \times Base + S$

La definizione generale data qui sopra è un numero a sette digit; qualsiasi altro numero di digit potrà essere rappresentato aggiungendo digit all'estrema sinistra del numero.

Ogni volta che si moltiplica "$base^n$ x base", si ottiene $base^{n+1}$; cosicché da "base" a "$base^6$" si hanno i seguenti valori:

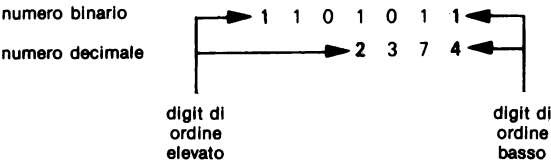
binario		decimale
due	base	dieci
quattro	$base^2$	cento
otto	$base^3$	mille
sedici	$base^4$	diecimila
trentadue	$base^5$	centomila
sessantaquattro	$base^6$	un milione

DIGIT DI ORDINE ELEVATO
DIGIT DI BASSO ORDINE

In ogni numero multidigit ci riferiamo ai digit uno, cioè al digit più a destra, come al digit di "basso ordine". Il digit più a sinistra è chiamato digit di "ordine elevato". Tutto ciò può essere illustrato come segue:

numero binario 1 1 0 1 0 1 1

numero decimale 2 3 7 4

digit di ordine elevato — digit di ordine basso

CONVERSIONE BINARIO/DECIMALE

Si può usare la rappresentazione generale di un numero multidigit per convertire qualsiasi numero binario al suo equivalente decimale. **Riportiamo qui alcuni esempi di numeri binari multidigit che mostrano come raffigurare i loro equivalenti decimali.**

digit otto

digit quattro

digit due

digit uno

1 0 1 1

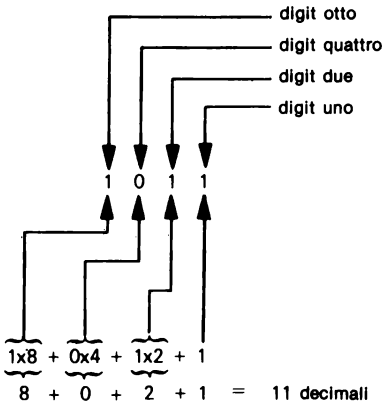
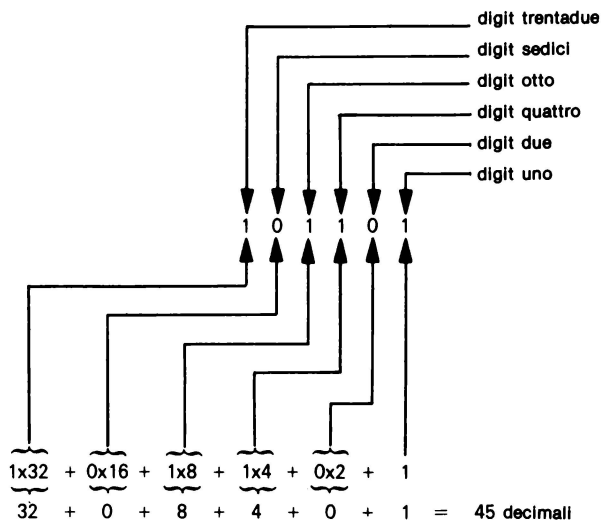
1x8 + 0x4 + 1x2 + 1

8 + 0 + 2 + 1 = 11 decimali

digit trentadue
digit sedici
digit otto
digit quattro
digit due
digit uno

1 0 1 1 0 1

1x32 + 0x16 + 1x8 + 1x4 + 0x2 + 1

32 + 0 + 8 + 4 + 0 + 1 = 45 decimali

CONVERSIONE DECIMALE/BINARIO

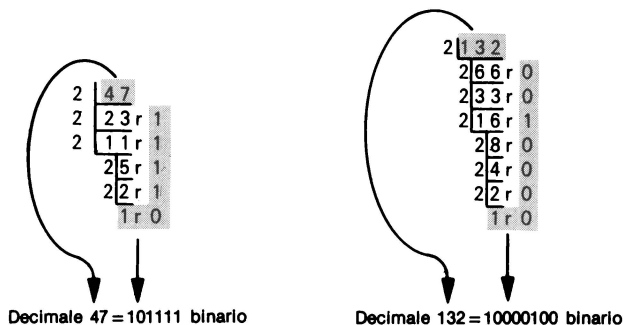
Esiste una tecnica molto semplice per convertire ciascun numero decimale al suo equivalente binario; definiremo semplicemente questa tecnica, poi spiegheremo perché funziona.

C'è una definizione di questa tecnica: per convertire un numero decimale nel suo equivalente binario, dividete ripetutamente il numero decimale per due, finché nulla è rimasto del numero. Qui abbiamo due esempi.

2 | 47
2 | 23 r 1
2 | 11 r 1
2 | 5 r 1
2 | 2 r 1
1 r 0

Decimale 47 = 101111 binario

2 | 132
2 | 66 r 0
2 | 33 r 0
2 | 16 r 1
2 | 8 r 0
2 | 4 r 0
2 | 2 r 0
1 r 0

Decimale 132 = 10000100 binario

Ora spiegheremo perché questa tecnica di conversione funziona. I passaggi illustrati qui sopra forniscono l'equivalente binario multidigit del numero decimale; si ottiene per primo il digit binario meno significativo (cioè, il più a destra). Questo digit è il resto una volta saputo quanti digit due ci sono.

Usiamo il simbolo NNN per rappresentare qualsiasi numero decimale. Cosa avviene quando dividiamo NNN per due? Otterremo la metà di NNN, più un resto di 0 o 1. Usiamo il simbolo PPP per rappresentare la metà di NNN; nel caso generale è così che illustriamo il fatto che NNN è diviso per 2.

```
2 | NNN
    PPP      resto      0 o 1
```

Qui vi sono alcuni casi specifici:

```
2 | 421                (NNN = 421)
    210      resto 1   (PPP = 210)

2 | 36                 (NNN = 36)
    18       resto 0   (PPP = 18)

2 | 7                  (NNN = 7)
    3        resto 1   (PPP = 3)
```

In ciascuna di queste illustrazioni il numero decimale NNN è mostrato come costituito da PPP digit due, più 0 o 1:

NNN	=	PPP	x	2	+	resto
421	=	210	x	2	+	1
36	=	18	x	2	+	0
7	=	3	x	2	+	1

Per ciascun numero decimale NNN, allo scopo di scoprire quanti digit due ci sono (PPP), dividete semplicemente il numero decimale (NNN) per due. Il resto (0 o 1) è il digit degli uno.

Cosa dire di PPP? E' un numero decimale. Se per caso PPP è zero o 1, allora si tratta anche di un vero numero binario; qualunque numero superiore non è un vero numero binario.

Se il numero di digit due calcolato nel primo passaggio di cui sopra (PPP) è più di 1, ciò significa che nel numero binario ci sono alcuni digit quattro. Per calcolare quanti digit quattro ci sono, potete semplicemente divider il numero iniziale decimale per quattro.

```
4 | NNN
    QQQ      resto  3,2,1
    o 0
```

Vediamo ancora gli esempi precedenti.

NNN	=	QQQ	x	4	+	resto
421	=	105	x	4	+	1
36	=	9	x	4	+	0
7	=	1	x	4	+	3

Ma notiamo che QQQ, il numero dei digit quattro, deve essere metà di PPP, il numero di digit due; il che è come dire che dividere NNN per quattro è lo stesso che dividere metà di NNN per due.

```
4 | NNN                      è lo stesso che   2 | NNN
    QQQ , resto 3, 2, 1 o 0                    2 | PPP   resto 1 o 0
                                                   QQQ   resto 1 o 0

4 | 421                      è lo stesso che   2 | 421
    105   resto 1                              2 | 210   resto 1
                                                   105   resto 0

4 | 36                       è lo stesso che   2 | 36
     9    resto 0                              2 | 18    resto 0
                                                    9    resto 0

4 | 7                        è lo stesso che   2 | 7
    1     resto 3                              2 | 3     resto 1
                                                   1     resto 1
```

Il vantaggio di dividere due volte per due è che tutti i resti sono 0 o 1, precisi digit binari, esaminiamo quei resti riprodotti qui sopra:

1	**è lo stesso che**	**01**	**(binario)**
0	**è lo stesso che**	**00**	**(binario)**
3	**è lo stesso che**	**11**	**(binario)**

Il decimale (7_{10}) è diventato il binario 111_2; cioè il decimale sette equivale ad un digit quattro, più un digit due, più un digit uno:

$$7_{10} = 4_{10} + 2_{10} + 1_{10}$$

I numeri decimali 36_{10} e 421_{10} devono continuare ad avere digit binari a livello più alto perchè 9_{20} e 105_{10} non sono veri numeri binari. Digit binari a livelli più alti si ottengono continuando a dividere per due; qui c'è la conversione completa di 36.

```
2 | 36
2 | 18    resto 0 (nessun digit uno)
   2 | 9  resto 0 (nessun digit due)
   2 | 4  resto 1 (nessun digit quattro)
   2 | 2  resto 0 (nessun digit otto)
       1  resto 0 (nessun digit sedici)
       ↑
       1  32   digit
```

Talché $36_{10} = 32_{10} + 4_{10} = 100100_2$.

NOTAZIONE DECIMALE

Nelle precedenti illustrazioni abbiamo introdotto una forma stenografica che si usa comunemente nei libri sui computer. **I numeri decimali vengono identificati da una appendice posta alla fine del numero in basso:**
il decimale 4713 è rappresentato come 4713_{10}

NOTAZIONE BINARIA

I numeri binari vengono identificati da un'appendice alla fine del numero in basso:
il binario 11010 è rappresentato come 11010_2.
Poiché il numero decimale è stato ripetutamente diviso per il numero decimale 2, il resto può essere solamente 0 o 1; il resto inoltre dice quanti digit uno, digit due, digit quattro e così via, ci sono nell'equivalente binario del numero decimale.

Se QQQ è 2 o più, allora ci sono più di 0 o 1 digit quattro, e dividerete i digit quattro (QQQ) per due per determinare quanti digit otto ci sono; il resto, quando dividete i digit quattro (QQQ) per due, vi dice se ci sono 0 o 1 digit quattro. Se c'è più di un digit otto, allora passate al digit sedici, e se c'è più di un digit·sedici, passerete al digit trentadue, e così via.

Tab. 4-1 - Tutti i numeri binari a quattro digit e la loro rappresentazione decimale.

Numeri decimali	Numeri binari
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Tab. 4-2 - Il numero più alto che si può rappresentare mediante numeri binari con digit da 1 a 16

Numero di digit binari	Massimo valore binario	Equivalente decimale
1	1	1
2	11	3
3	111	7
4	1111	15
5	11111	31
6	111111	63
7	1111111	127
8	11111111	255
9	111111111	511
10	1111111111	1023
11	11111111111	2047
12	111111111111	4095
13	111111111111	8191
14	11111111111111	16383
15	111111111111111	32767
16	1111111111111111	65535

Da notare che ogni volta che viene aggiunto un digit binario si raddoppia il numero massimo che può essere rappresentato.

BIT, NIBBLE E BYTE

BIT

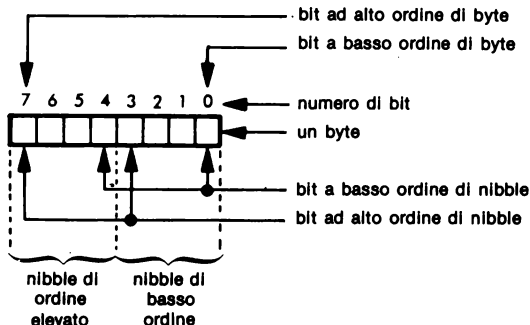
Un digit binario viene sempre indicato con l'abbreviazione bit, quindi un bit può avere valori da 0 a 1.

BYTE

Quantunque si possano creare numeri da qualsiasi numero di digit binari, come illustrato in tab. 2, vi sono certi numeri di bit che incontrerete frequentemente; in particolare avrete spesso a che fare con combinazioni di 8 bit. **Un'unità di 8 bit è indicata come byte.**
Esistono alcuni rari tipi di computer che usano la parola byte per descrivere alcuni altri numeri di bit (per esempio, 6), ma nel mondo dei microcomputer, il byte è sempre un'unità da 8 bit. Cosicché un byte può rappresentare numeri compresi fra 0 e 255.

NIBBLE

Talvolta le unità da 4 bit sono indicate come nibble: cosicché un byte consiste in due nibble.

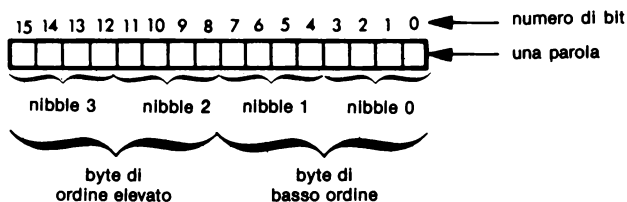
bit ad alto ordine di byte
bit a basso ordine di byte
7 6 5 4 3 2 1 0 numero di bit
un byte
bit a basso ordine di nibble
bit ad alto ordine di nibble
nibble di ordine elevato
nibble di basso ordine

WORD

Le unità da 16 bit vengono spesso chiamate word (o parole). Cosicché una word consiste in due byte o in quattro nibble.

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 ← numero di bit

← una parola

nibble 3 nibble 2 nibble 1 nibble 0

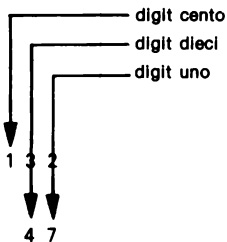
byte di ordine elevato — byte di basso ordine

ARITMETICA BINARIA

Vediamo i paralleli fra l'aritmetica che usa numeri binari e quella che usa numeri decimali.

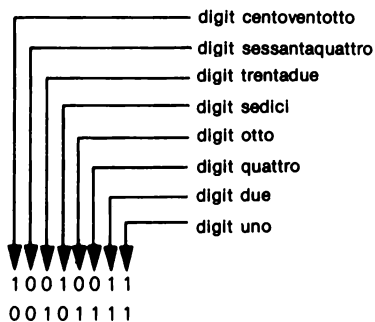
SOMMA BINARIA

Quando effettuate l'addizione decimale, allineate i digit di due numeri da sommare come segue:

digit cento
digit dieci
digit uno

1 3 2
4 7

Per l'addizione binaria, voi fate essenzialmente la stessa cosa:

digit centoventotto
digit sessantaquattro
digit trentadue
digit sedici
digit otto
digit quattro
digit due
digit uno

1 0 0 1 0 0 1 1
0 0 1 0 1 1 1 1

Voi sommate i numeri, un digit alla volta, partendo con i digit di basso ordine (uno). Ma l'addizione binaria è ben semplice, se paragonata a quella decimale. Quando sommate due digit binari, ci sono ben quattro possibilità:

1 ← portare 1

```
  0     0     1     1
+ 0   + 1   + 0   + 1
 --    --    --    --
  0     1     1     0
```

Quando avete due digit decimali, avete 100 possibilità, 45 delle quali genereranno un carry. **Esaminiamo ora alcuni esempi di somma binaria;** e prima di tutto il semplicissimo 2 + 2 = 4. Usando numeri binari, ciò è quanto si ottiene.

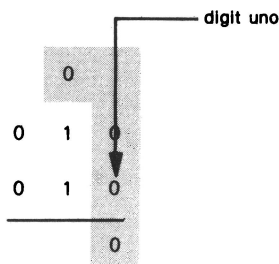
Decimali	Binari
0	000
1	001
2	010
3	011
4	100

Decimale	Binario
2	010
+ 2	+ 010
= 4	= 100

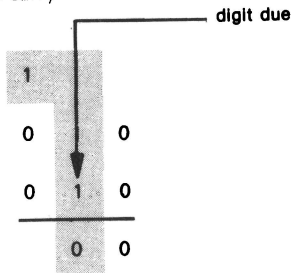
Vediamo una spiegazione digit per digit dell'addizione binaria.

1) I digit uno sono ambedue zero; sommandoli si origina uno zero e nessun carry.

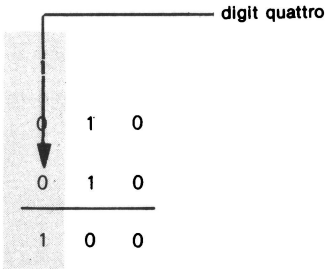
digit uno

```
      0
0  1  0
0  1  0
-------
      0
```

2) I digit due sono ambedue 1 e non c'è alcun carry dai digit uno. Sommando i due bit 1, si crea uno 0 più un carry·

digit due

```
1
0  1  0
0  1  0
-------
   0  0
```

3) I digit quattro sono ambedue 0, ma c'è un carry dai digit due. I due zeri danno 0, ma sommando 1 (il carry) a 0 si ottiene 1, senza carry.

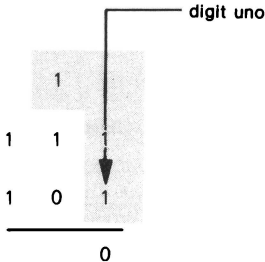
digit quattro

```
1
0 1 0
0 1 0
-----
1 0 0
```

Vediamo ora un esempio leggermente più complesso di somma binaria: 7 + 5 = 12. Questo si può illustrare come segue:

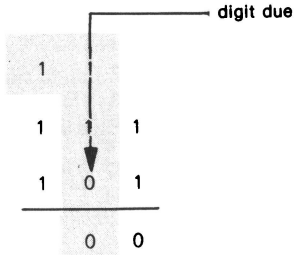
Decimale	Binario
7	111
+ 5	+ 101
= 12	= 1100

I digit uno sono ambedue 1; essi si sommano in 0 e danno un carry:

digit uno

```
  1
1 1 1
1 0 1
-----
    0
```

I digit due sono 1 e 0; c'è anche un carry dai digit uno. Ciò equivale a sommare i tre bit; 1,1 e 0, da cui si ottiene 0 e un carry:

digit due

```
1 1
1 1 1
1 0 1
-----
  0 0
```

I digit quattro sono ambedue 1, ma c'è anche un carry dai digit due. Sommare tre bit 1 è abbastanza semplice se si fa in due passaggi. Prima si sommano due bit 1

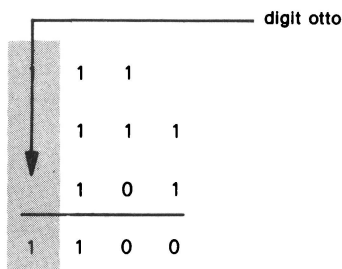
$$\begin{array}{r} 1 \\ 1 \\ \hline 10 \end{array}$$

Questo dà 0 ed un carry; ora si aggiunge il terzo bit al risultato:

$$\begin{array}{r} 10 \\ 1 \\ \hline 11 \end{array}$$

Si ottiene 1 con un carry. I digit otto sono ambedue 0, ma c'è un carry, perciò i digit otto sommati danno 1.

digit otto

$$\begin{array}{cccc} & 1 & 1 & \\ & 1 & 1 & 1 \\ & 1 & 0 & 1 \\ \hline 1 & 1 & 0 & 0 \end{array}$$

La somma di 132 e 47, che abbiamo illustrato precedentemente descrivendo la logica del microcomputer in binario diventa:

Decimale	Binario
132	10000100
+ 47	00101111
= 179	10110011

Le conversioni da decimale a binario di 132 e 47 sono già state descritte. **Possiamo controllare che la somma binaria è realmente equivalente al decimale 179 come segue:**

10110011

1	x	1	=	1
1	x	2	=	2
0	x	4	=	0
0	x	8	=	0
1	x	16	=	16
1	x	32	=	32
0	x	64	=	0
1	x	128	=	128
				179

Se a questo punto non avete capito il metodo binario di fare addizioni, dovrete continuare a fare esempi finché non abbiate "digerito" la somma binaria.

SOTTRAZIONE BINARIA E NUMERI NEGATIVI

La sottrazione binaria è molto più semplice di quella decimale, in quanto si hanno solo quattro possibilità.
Potete sottrarre 0 da 0, che dà 0

$$\begin{array}{r} 0 \\ -0 \\ \hline 0 \end{array}$$

Potete sottrarre 1 da 1, che dà ancora 0

$$\begin{array}{r} 1 \\ -1 \\ \hline 0 \end{array}$$

Se sottraete 0 da 1 il risultato è 1

$$\begin{array}{r} 1 \\ -0 \\ \hline 1 \end{array}$$

Ma cosa accade quando sottraete 1 da 0? Esattamente come fareste per la sottrazione decimale, così per la sottrazione binaria voi dovete prendere un prestito (borrow) dal bit più alto successivo come segue:

Borrow

$$\begin{array}{r} 10 \\ -1 \\ \hline 1 \end{array}$$

10 è la rappresentazione binaria di due; l'illustrazione qui sopra sottrae 1 da 2, lasciando un risultato 1. Se non c'è un bit più alto da cui prendere un borrow, allora il risultato è –1:

$$\begin{array}{r} 0 \\ -1 \\ \hline -1 \end{array}$$

Estendendo la sottrazione ai numeri multi bit (digit binari multipli) è semplice esattamente come l'estendere la sottrazione a numeri a digit decimali multipli.
Qui c'è l'equivalente binario di 4-2 = 2

$$\begin{array}{r} 100 \\ 010 \\ \hline 010 \end{array}$$

SOTTRAENDO
DIMINUENDO

Quando sottraete due numeri, voi sottraete un sottraendo da un diminuendo. Ciò si può illustrare come segue:

```
  4 ◄—— Diminuendo
- 2 ◄—— Sottraendo
= 2 ◄—— Differenza
```

Esaminando la sottrazione binaria di due da quattro, i digit uno sono ambedue 0; quindi la differenza è 0.

```
          digit uno
1  0  0 ◄—— diminuendo
0  1  0 ◄—— sottraendo
      0
```

Nel caso dei digit due, dobbiamo sottrarre 1 da 0. Perciò noi prendiamo un borrow 1 dal digit quattro diminuendo, e otteniamo la differenza di 1.

```
                        digit due
1  0  0      0  10  0
         ——►
0  1  0      0  1   0
                1   0
```

Il diminuendo digit quattro è ora 0; l'1 che c'era è stato preso a borrow dal diminuendo digit due. Così per i digit quattro noi sottraiamo 0 da 0, creando una differenza di 0:

```
digit quattro
0  10  0
0  1   0
0  1   0
```

Andando ad un esempio più complesso, diamo qui la rappresentazione binaria di $132_{10} - 47_{10} = 85_{10}$.

Decimale	Binario	
11	111	◄—— Borrow
132	10000100	◄—— Diminuendo
- 47	- 00101111	◄—— Sottraendo
= 85	01010101	◄—— Differenza

Diamo una rappresentazione passo-passo della sottrazione binaria.

```
1 0 0 0 0 1 0 0          1 0 0 0 0 0 1 10
0 0 1 0 1 1 1 1   --->   0 0 1 0 1 1 1 1
                         ---------------
                                       1

1 0 0 0 0 0 1 10         0 1 1 1 1 10 1 10
0 0 1 0 1 1 1 1   --->   0 0 1 0 1 1 1 1
---------------          ---------------
            0 1                    1 0 1

0 1 1 1 1 10 1 10
0 0 1 0 1 1 1 1
---------------
0 1 0 1 0 1 0 1
```

Possiamo controllare che il risultato binario è corretto generando il suo equivalente decimale.

01010101

1	x	1	=	1
0	x	2	=	0
1	x	4	=	4
0	x	8	=	0
1	x	16	=	16
0	x	32	=	0
1	x	64	=	64
0	x	128	=	0
				85

Ora, ciò potrà sorprendervi, ma molti computer non possono far sottrazioni; essi posson solo fare addizioni. Fortunatamente questo non è un problema: sottrarre un numero da un altro è equivalente a sommargli il negativo del numero. Ciò si può illustrare come segue:

$$4 + - (2) = + (-2)$$

Questa soluzione potrebbe sembrare ridicola, ma in effetti non lo è; sembra ridicola solo perchè siete abituati ai numeri decimali.
Consideriamo la sottrazione di 3 da 9. Potremo scrivere questo come 9-3, o come 9 + (0—3), che in effetti è più complicato.
Per risolvere 9 + (0—3), si deve sottrarre 3 da 0 prima di aggiungere il risultato a 9; ma quando si sottrae 3 da 0, si ottiene —3, che mette ancora a posto il fatto di sottrarre 3 da 9.
La sottrazione nel mondo degli interruttori e dei numeri binari non può essere risolta finché non troviamo un sistema per rappresentare i numeri negativi.
Un interruttore è un dispositivo a due stati, e non possiamo semplicemente aggiungere un nuovo stato perché l'interruttore ci rappresenti un segno negativo:

0   1   ? —

Quindi finché non abbiamo trovato qualche mezzo di rappresentare numeri binari negativi, non possiamo nemmeno tentare di venirne fuori dall'equivalente binario di 9 + (0−3).

Allo scopo di trovare qualche metodo per rappresentare i numeri binari negativi, cominciamo col semplice caso di numeri compresi nella gamma fra 0 e −7.

Nella loro forma positiva questi numeri sono rappresentati da tre digit binari:

decimale	equivalente binario
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Noi non possiamo scegliere arbitrariamente un metodo di rappresentazione dei numeri binari negativi; la nostra esigenza è di poter sottrarre aggiungendo la rappresentazione negativa del numero.

Il sistema logico di trovare una rappresentazione binaria dei numeri negativi è di tentare di sottrarre il numero positivo da zero. Consideriamo +3; la sua forma binaria è 11_2. Vediamo cosa succede quando sottraiamo 11_2 da 00.

```
  00  ◄—— diminuendo
- 11  ◄—— sottraendo
 =?
```

Partendo coi digit uno, noi abbiamo da sottrarre 1 da 0; questo è impossibile, cosicché noi tentiamo il borrow di 1 diminuendo digit due, che è pure 0. Se assumiamo di non poter fare il borrow di 1 dal diminuendo digit quattro (alla sinistra del digit due), allora ciò è quanto otteniamo:

```
┌──────── digit due borrow 1
│ ┌────── digit due
│ │ ┌──── digit uno
▼ ▼ ▼
1 0 0  ◄—— diminuendo
- 1 1  ◄—— sottraendo
 = ?
```

Ora il diminuendo digit uno può fare il borrow di 1 dal diminuendo digit due.

```
┌──────── digit due (10-1 =1)
│ ┌────── digit uno borrow 1
│ │ ┌──── digit uno
▼ ▼ ▼
1 1 0  ◄—— diminuendo
- 1 1  ◄—— sottraendo
=   ?
```

Possiamo allora effettuare con successo la sottrazione:

```
┌──────────── digit due
│  ┌───────── digit uno
▼  ▼
 1 10  ◄──── diminuendo
-1  1  ◄──── sottraendo
 ────
 0  1  ◄──── differenza (negativa)
```

La differenza nel digit uno è calcolata come $10_2 - 1_2 = 1_2$. Questo è l'equivalente del decimale $2_{10} - 1_{10} = 1_{10}$.

La differenza nei digit due è semplicemente 1—1=0. Noi però abbiamo ottenuto un successo nel sottrarre 3 da 0 usando l'aritmetica binaria, ma ciò ha voluto dire un gioco di prestigio del quale dobbiamo ora tener conto; noi abbiamo preso il borrow di 1 dal digit successivo di ordine elevato del diminuendo (il digit quattro in questo caso) quando non esiste alcun digit di tale tipo.

Abbiamo ancora un problema che ora richiede soluzione: i due bit 01 sono mostrati come rappresentanti il valore —3; ma essi rappresentano pure il valore +1.

Questi due problemi non hanno alcuna soluzione nel contesto del calcolo binario come lo abbiamo sin qui definito.

Allo scopo di poter maneggiare la sottrazione — e gli inevitabili numeri negativi che ne possono risultare — noi dobbiamo modificare le regole sin qui adottate per il calcolo binario.

Ma la nuova serie di regole deve essere logicamente e numericamente pertinente con la necessità di numeri binari positivi e con le necessità dell'aritmetica binaria.

Fortunatamente, c'è una soluzione semplice. Vediamo alcuni numeri decimali col loro segno:

+ 10	+ 123	+ 47	+ 83742
- 10	- 123	- 47	- 83742

Il fatto nuovo introdotto dai numeri qui illustrati è il segno; c'è un segno più (+) ed un segno meno (-).

I numeri binari col segno potrebbero di conseguenza essere illustrati come segue:

+ 1011	+ 1110101	+ 110	+ 1011010
- 1011	- 1110101	- 110	- 1011010

IL SEGNO DEI NUMERI BINARI

Quello che abbiamo è un segno più (+) o meno (-) che precede la fila dei bit (digit binari). Ora, non possiamo rappresentare i segni più e meno come entità separate e distinte entro la logica del computer; ricordiamo che la logica del computer consiste di null'altro che commutatori a due stati.

Dobbiamo perciò prendere lo stesso commutatore a due stati che rappresenta i digiti 0 e 1, ma ora dobbiamo usarlo per rappresentare i segni più e meno.

Se usiamo un commutatore disinserito ("off") per rappresentare un segno più, ed un commutatore inserito ("on") per rappresentare un segno meno, allora il segno più (+) ed il digit zero (0) sono ambedue rappresentati da un commutatore in posizione "off"; il segno meno (-) e il digit uno (1) sono ambedue rappresentati da un commutatore in posizione "on".

Ora i nostri numeri binari col segno potrebbero essere rappresentati come segue:

01011	01110101	0110	01011010	numeri positivi
11011	11110101	1110	11011010	numeri negativi

Disgraziatamente, il metodo di rappresentazione dei numeri negativi illustrato qui sopra non è destinato a funzionare.

Consideriamo il semplice esempio: 5-3=2. Ricordiamo: abbiamo detto che, allo scopo di fare sottrazioni, dobbiamo sommare la rappresetanzione negativa del numero.

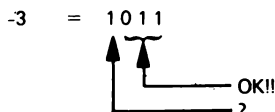
Così 5-3=2 diventa 5+(-3)=2 Ma questo funziona? Proviamo a vedere:

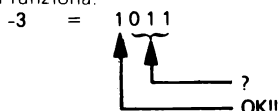
Se +5=0101 + 3=0011 e −3=1011

allora +5+(−3) diventa

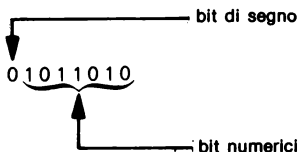
```
  0101
+ 1011
 10000
```

Non funziona. O il metodo che abbiamo adottato per rappresentare il segno non funziona.

```
-3  =  1011
       ↑ ↑
       | └──── OK!!
       └────── ?
```

O è il metodo che abbiamo adottato per rappresentare la porzione numerica del numero negativo che non funziona.

```
-3  =  1011
       ↑ ↑
       | └──── ?
       └────── OK!!
```

Per riuscire a trovare ciò che è sbagliato esaminiamo più accuratamente i numeri negativi. Il solo modo in cui possiamo dire se un digit binario rappresenta il segno di un numero o un digit entro il numero è nel guardare alla posizione del digit binario.

La posizione più a sinistra del bit deve essere interpretata come il bit del segno:

```
┌──────────── bit di segno
↓
0 1011010
     ↑
     └─────── bit numerici
```

Come farete a dire la differenza che esiste fra un numero binario col segno ed un numero binario senza segno che sia 1 digit più lungo?

La risposta è che non potete dire la differenza con una semplice occhiata; dovete semplicemente conoscere quello con cui avete a che fare.

Questa necessità di interpretare i numeri non è naturalmente nuova per noi; abbiamo già discusso l'interpretazione dei numeri nel 2° capitolo, a proposito del programma di pagamento di Joe Bitburger.

Dicendo che dovete sapere in anticipo se un numero binario ha o non ha il segno è più o meno lo stesso che dire che bisogna far differenza fra un ammontare in dollari ed un numero di serie su un assegno. Questo tipo di interpretazione numerica è una necessità costante quando si ha a che fare coi microcomputer.

Occorre far differenza fra i dati numerici col segno e quelli senza segno, più molti sistemi addizionali nei quali può capitare di interpretare una sequenza di bit; è ciò che fa nascere la potenza e la complessità della programmazione di un computer.

Ritornando ai numeri binari col segno, come li abbiamo definiti, esaminiamo in modo critico il bit del segno.

Scegliendo il commutatore "off" per rappresentare o il segno più o un bit 0, i numeri binari con segno positivo hanno la stessa interpretazione numerica dei numeri binari senza segno. Ciò può essere illustrato come segue:

numero decimale		**numero binario**
3_{10}	=	11_2
$+3_{10}$	=	011_2

Quando passiamo da numeri positivi senza segno a numeri positivi con segno, aggiungiamo uno 0 iniziale, che non influisce in alcun modo sull'ampiezza del numero. Ciò significa che la nostra rappresentazione del segno è soddisfacente.

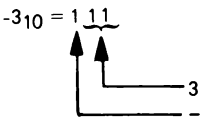
Cosa possiamo dire sui bit numerici?

Abbiamo una scelta fra due soluzioni:

1) possiamo piazzare un 1, che rappresenta il segno negativo, alla sinistra del bit a ordine elevato di numero positivo

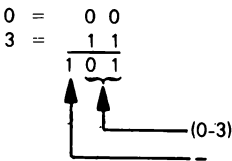
$$-3_{10} = 1\,\underbrace{11}$$

3

-

2) alternativamente possiamo prendere il gruppetto di digit binari creato sottraendo il numero positivo da 0, come abbiamo già fatto sottraendo 3 da 0, e possiamo piazzare l'uno alla sinistra del bit numerico più significativo:

$$\begin{array}{rcr} 0 & = & 0\;0 \\ 3 & = & \underline{1\;1} \\ & & 1\;\underbrace{0\;1} \end{array}$$

(0-3)

-

Succede così che il metodo (1), che avevamo scelto automaticamente, non funziona (come abbiamo dimostrato tentando di sottrarre 3 da 5).

Il metodo (2) costituisce la via più semplice. L'uso del metodo (2) è come noi useremmo quattro digit binari per rappresentare i numeri nella gamma da 0 a 7:

numeri decimali		**numeri binari**	
Positivo	**Negativo**	**Positivo**	**Negativo**
0	0	0000	0000
1	1	0001	1111
2	2	0010	1110
3	3	0011	1101
4	4	0100	1100
5	5	0101	1011
6	6	0110	1010
7	7	0111	1001

bit di numeri

bit di segno

COMPLEMENTO DUE

La rappresentazione di numeri negativi qui sopra illustrata è indicata come il "complemento due" del numero.

FORMAZIONE DEL COMPLEMENTO DUE

COMPLEMENTO UNO

Per generare i complementi due di un numero, non si deve sottrarre il numero positivo da 0 e poi aggiungergli un bit 1 di ordine elevato (il più a sinistra) per rappresentare il segno meno; esiste una procedura più semplice. **Prima si forma il complemento uno del numero** invertendo ciascun bit, vale a dire rimpiazzando i bit 0 con gli 1, e i bit 1 con gli 0.

Riportiamo qui qualche esempio di numeri binari e dei loro complementi uno.

numero binario:	101	11101	101010
complementi uno:	010	00010	010101

Aggiungete 1 al complemento uno del numero e ne avrete il complemento due. Riportiamo alcuni esempi, da cui potete verificare in confronto ai numeri binari negativi sopra riportati:

+3	=	0011		
complementi uno	=	1100		
complementi due	=	1101	=	-3
+5	=	0101		
complementi uno	=	1010		
complementi due	=	1011	=	-5

Vediamo ora se questa rappresentazione binaria di numeri negativi è valida, se è così, allora **dobbiamo essere in grado di sottrarre un numero aggiungendo il suo complemento due, cioè la sua rappresentazione negativa.**

Considериamo i numeri nella gamma da 1 a 7; se siamo sicuri di sottrarre un numero più piccolo da un numero più grande, allora **qui abbiamo alcuni esempi che verificano che abbiamo sviluppato una rappresentazione valida per i numeri negativi:**

decimale	binario
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

7-3 = 4

diventa 7 + (-3) = 4

  0 1 1 1
+1 1 0 1
= 1 0 1 0 0

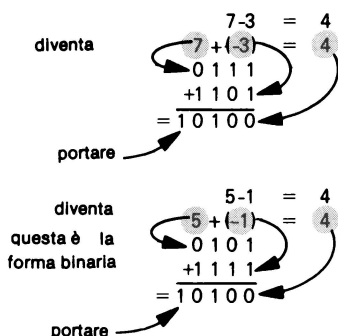
portare

5-1 = 4

diventa 5 + (-1) = 4

questa è la forma binaria

  0 1 0 1
+1 1 1 1
= 1 0 1 0 0

portare

Il risultato è valido, ma c'è un carry. Ogni qualvolta si genera per sottrazione una differenza positiva, c'è un carry di 1.

Cosa succede se si sottrae un numero più grande da uno più piccolo? Esattamente la stessa cosa che avviene con la sottrazione decimale: **si rimane con un numero negativo.**

Qui riportiamo a titolo di esempio, cosa succede sottraendo 5 da 3:

	3-5	=	-2
diventa	3 + (-5)	=	+(-2)
questa è la forma binaria	0 0 1 1		
	+1 0 1 1		
	= 1 1 1 0		

Siete in grado di dire che il risultato è negativo perché il digit di ordine elevato del risultato binario è 1.

Ricordate che, quando avete a che fare con numeri binari positivi e negativi, il bit di ordine elevato rappresenta il segno del numero: se tale bit è 0, il numero è positivo; se tale bit è 1 (come sopra), allora il numero è negativo.

E' molto facile creare l'equivalente positivo di un numero negativo; basta prendere il complemento due del numero negativo e si ha di nuovo il numero positivo.

Indichiamo alcuni esempi di numeri positivi, dei loro complementi due equivalenti negativi, e della rigenerazione del numero positivo originale.

+7	=	0111		
complementi uno	=	1000		
complementi due	=	1001	=	-7
complementi uno	=	0110		
complementi due	=	0111	=	+7
+4	=	0100		
complementi uno	=	1011		
complementi due	=	1100	=	-4
complementi uno	=	0011		
complementi due	=	0100	=	+4

Non c'è nulla di particolarmente astuto nel prendere due volte il complemento due di un numero e riavere così il numero. Dopo tutto, nel campo dell'aritmetica decimale due negativi fanno un positivo; per esempio $-(-2)$ è uguale a $+2$.

Così, se la rappresentazione binaria che abbiamo sviluppato per i numeri negativi è valida, allora il complemento due di un numero negativo deve ridare il numero positivo, cosa che esso fa.

Abbiamo sviluppato un metodo molto elegante di manipolare numeri negativi e di far sottrazioni usando digit binari; ma ricordate, i computer non possiedono l'abilità intrinseca di cimentarsi con numeri negativi.

Per tutto il tempo che usate un computer, è nella vostra responsabilità ricordare se una tabella di digit binari rappresenta solo numeri positivi, o contemporaneamente numeri positivi e negativi.

MOLTIPLICAZIONE E DIVISIONE BINARIA

Non ci addentreremo molto in dettaglio a discutere la moltiplicazione e la divisione binaria; la conoscenza di questo problema vi sarà utile solo quando avrete una maggiore esperienza nell'uso dei microcomputer.

C'è comunque, associato con la moltiplicazione e la divisione dei numeri binari, un fenomeno molto interessante. Allo scopo di moltiplicare un numero binario per due, voi spostate ciascun bit di un posto verso sinistra, come nell'esempio:

$$27_{10} = 1B_{16} = 00011011_2$$
$$27_{10}\times 2 = 54_{10} = 36_{16} = 00110110_2$$

Spostare ciascun bit di un numero binario di una posizione verso destra, equivale a dividere il numero per due, come nell'esempio.

$$36_{10} = 24_{16} = 00100100_2$$
$$18_{10} = 12_{16} = 00010010_2$$

In realtà, non c'è nulla di molto sorprendente nello spostare digit binari per moltiplicare o dividere per due; potete fare la stessa cosa coi numeri decimali. Per moltiplicare un numero decimale per dieci, voi spostate ciascun digit di una posizione a sinistra.

$$374 \times 10 = 3740$$

Spostando i digit decimali di una posizione a destra, si divide per dieci:

$$\frac{26730}{10} = 2673$$

Esiste una varietà di sistemi escogitati per moltiplicare e dividere numeri binari; programmando un microcomputer in un linguaggio a più alto livello, non capiterà mai di aver a che fare con l'esatta procedura con la quale vengono moltiplicati i numeri binari: ci penserà il compilatore. Perchè mai qualcuno dovrebbe stare a seccarsi con il linguaggio di assembly? La risposta ancora una volta è per avere un miglior controllo del vostro programmatore.

Per esempio, ci sono molti programmi diversi che si possono scrivere per ottenere la moltiplicazione o la divisione binaria. I vari programmi di moltiplicazione e divisione vi daranno tutti lo stesso risultato; ma qualcuno di loro esegue molto rapidamente mentre richiede un sacco di memoria per immagazzinare il programma, mentre altri richiedono un lungo tempo per eseguire ma hanno programmi relativamente corti.

Se usate un linguaggio di livello più elevato, potrete ottenere qualunque programma di moltiplicazione o divisione che possa capitare. Se scrivete il vostro programma, potete scegliere il metodo di moltiplicazione o divisione che sia il più veloce da eseguire, o il metodo che usi la minor memoria.

NUMERI OTTALI ED ESADECIMALI

Ci vuole un genio della matematica per convertire, con una semplice occhiata, da decimali multidigit a numeri binari; questo tipo di conversione non è certo facile.
Questo però è un problema per gli uomini, ma non per un microcomputer. Un microcomputer lavora interamente con digit binari; non sa nemmeno che esistono digit decimali, mentre gli uomini trovano molto difficile operare coi numeri binari. **La manipolazione di numeri binari non è conveniente, fa perdere tempo ed è incline ad errori.** Immaginate quanto è facile scambiare un 1 con uno 0, e come sarà difficile scoprire questo errore.
Ciò ha dato come risultato che sono stati adottati sistemi di conteggio che sono più concisi del binario, ed hanno inoltre una relazione semplice con i digit binari. I due sistemi numerici più comunemente usati sono l'"ottale" e l'"esadecimale".

Ciascun digit ottale rappresenta esattamente tre digit binari, come segue:

binario:	000	001	010	011	100	101	110	111
ottale:	0	1	2	3	4	5	6	7

La parola "ottale" deriva dal numero 8, in quanto i numeri ottali sono a "base 8". Cosicché il numerale 10, che è dieci nel sistema di conteggio decimale, è otto nel sistema di conteggio ottale.
Potete osservare bene i digit ottali illustrati qui sopra e chiedervi che differenza c'è fra i digit ottali e quelli decimali. Non ce n'é alcuna nella gamma da 0 a 7, ma i digit ottali non hanno né 8 né 9. Cosicché, i digit multiottali e multidecimali non

avranno gli stessi valori. Ciò si illustra come segue:

decimale	ottale
5	5
6	6
7	7
8	10
9	11
10	12
11	13
12	14

I numeri ottali sono identificati da un 8 a seguire scritto in basso:

$$11_{10} = 13_8$$

11 (decimale) = 13 (ottale)

La conversione di numeri binari nei loro equivalenti ottali si effettua direttamente; basta semplicemente ripartire il numero binario in gruppi di tre digit e rimpiazzare ciascun gruppo di tre digit col suo digit ottale equivalente, come segue:

binario:	110	101	110	111
ottale:	6	5	6	7

Talché $110101\ 110111_2 = 6567_8$

Al contrario, se desiderate convertire un numero ottale nel suo equivalente binario, dovete semplicemente rimpiazzare ciascun digit ottale col suo equivalente binario a tre digit, come segue:

ottale:	2	5	7	4
binario:	010	101	111	100

$$2574_8 = 010101111100_2$$

Ciascun digit esadecimale rappresenta esattamente quattro digit binari come segue:

binario:	0000	0001	0010	0011	0100	0101	0110	0111
esadecimale:	0	1	2	3	4	5	6	7
decimale	0	1	2	3	4	5	6	7

binario:	1000	1001	1010	1011	1100	1101	1110	1111
esadecimale:	8	9	A	B	C	D	E	F
decimale:	8	9	10	11	12	13	14	15

Il termine esadecimale deriva dal numero sedici, in quanto i numeri esadecimali sono numeri in base sedici. Cosicché il numerale 10, che è dieci nel sistema di conteggio decimale ed otto in quello ottale, è sedici nel sistema di conteggio esadecimale. Ciò pone un nuovo problema: se 10 rappresenta sedici nel sistema esadecimale, allora in esso ci devono essere sedici digit numerici singoli, così come ci sono dieci digit numerici singoli nel sistema decimale.

I sei digit numerici addizionali sono rappresentati dalle lettere ABCDEF come sopra indicato. Bisogna quindi differenziare fra il caso delle lettere da A ad F che rappresentano digit esadecimali e il caso di semplici lettere dell'alfabeto.

Mentre può sembrare che tutto ciò renda le cose complicate senza necessità, in realtà vi accorgerete che ciò non costituisce mai un problema. Guardando un frammento di dati, capirete automaticamente se state esaminando numeri o testo e non sorgerà quindi alcuna confusione.

Dovreste avere ben chiaro che i sistemi ottale ed esadecimale sono utili all'uomo, ma i computer sono insensibili al sistema di conteggio usato per scrivere dati su un pezzo di carta: **essi riconoscono solamente i dati binari.**

Se i microcomputer non capiscono i sistemi ottale ed esadecimale, è realmente più facile per voi imparare sistemi di conteggio che restare coi numeri decimali e lottare con le conversioni binario-decimale? La risposta è sì: per voi sarà molto meglio imparare i sistemi di conteggio ottale ed esadecimale.

Illustriamo questo punto con un esempio. Il numero decimale 2735 ha i seguenti equivalenti binario, ottale ed esadecimale.

```
                      A     A     F     esadecimale
decimale 2735 =     1010  1010  1111    binario
                     5   2    5   7     ottale
```

Abbiamo descritto le tecniche standard che potete usare per creare digit binari da digiti decimali e viceversa. Ma queste tecniche fanno perder tempo, sono scomode e mai facili da lavorarci, non importa quanto bene abbiate capito i numeri binari e decimali.

D'altra parte, potete fare la conversione fra numeri ottali o esadecimali ed i loro equivalenti binari. Una volta che voi abbiate imparato a pensare in ottale o esadecimale, e veramente ciò è piuttosto semplice, avrete il vantaggio di una breve annotazione per scrivere dati e di un semplicissimo processo di conversione per andare dal binario agli equivalenti, e viceversa.

Queste argomentazioni sono sostanzialmente le stesse che stanno conducendo tutto il mondo al sistema di misura metrico. Non c'è alcuna differenza concreta fra il misurare le distanze in metri e chilometri, e misurarle in yard e miglia; similmente non c'è alcuna differenza concreta fra il contare in decimale o in esadecimale.

Ma in un caso le conversioni sono scomode, mentre nell'altro caso esse sono dirette. **La tabella 4-3 raccoglie i numeri nella gamma da zero a sedici, nella loro rappresentazione binaria, decimale, ottale ed esadecimale.**

Tabella 4-3 — Sistemi numerici.

esadecimale	decimale	ottale	binario
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
A	10	12	1010
B	11	13	1011
C	12	14	1100
D	13	15	1101
E	14	16	1110
F	15	17	1111
10	16	20	10000

L'unico aspetto dei numeri ottali ed esadecimali del quale dovrete interessarvi è la conversione fra questi numeri ed i numeri binari o esadecimali.
L'addizione e la sottrazione che usano numeri ottali ed esadecimali sono identiche all'addizione e sottrazione che usano numeri decimali, tenendo in mente, naturalmente, che ciascun sistema di numerazione ha la propria serie di digit numerici.

CONVERSIONE OTTALE-DECIMALE

Se desiderate convertire da ottale ad esadecimale o da esadecimale ad ottale, il sistema più semplice di farlo è mediante un passaggio intermedio binario, in quanto i numeri binari hanno una correlazione digit per digit sia coi numeri ottali che esadecimali.
Per esempio, consideriamo il numero esadecimale $3C2F_{16}$; possiamo produrre il suo equivalente ottale usando la tabella 4-3:

```
esadecimale:   3    C    2    F
binario:    0011 1100 0010 1111
ottale:     0 3    6    0   5  7
```

Così $3C2F_{16} = 36057_8$

Potete convertire il numero ottale 23754 nel suo equivalente esadecimale:

```
ottale:       2   3   7   5   4
binario:    010 011 111 101 100
esadecimale:  2   7     E   C
```

Così $23754_8 = 27EC_{16}$

CONVERSIONI DECIMALE-OTTALE E DECIMALE-ESADECIMALE

Convertendo un numero decimale al suo equivalente ottale od esadecimale si segue la logica che già abbiamo descritto per generare l'equivalente binario di un numero decimale.
Consideriamo prima di tutto la conversione di un numero decimale nel suo equivalente esadecimale. Facciamo la conversione iniziando dal digit meno significativo (cioè quello più a destra). Se il nostro numero esadecimale si avvia ad essere un numero a due digit, esso allora consisterà di 16 (la base) moltiplicato per un qualche digit fisso (R) con un resto di S:

$$RS_{16} = R_{10} \times 16_{10} + S_{10}$$

Per tirar fuori questo resto S, **dividiamo il numero decimale per la base (sedici) come segue:**

```
16 | NNN
     R resto S
```

NNN è un numero decimale, ed è uguale al numero esadecimale RS_{16}. Facciamo qui un esempio numerico reale:

```
16 | 124
     7 resto 12
```

Perciò 124 $= 7 \times 16 + 12$
così $124_{10} = 7C_{16}$

In tal modo, siamo in grado di generare i digit R ed S e disponiamo di una completa conversione decimale-esadecimale: il decimale 124 equivale all'esadecimale 7C.

Consideriamo ora un numero decimale più alto, e cioè 282_{10}; quando dividiamo 282_{10} per 16_{10}, facciamo questo:

16 | 282
17 resto 10

Il resto (S) è il decimale 10, che (dalla solita tabella) vediamo avere l'equivalente esadecimale A_{16}. Però R, il moltiplicatore per la base, ha l'equivalente decimale 17, e dalla tavola vediamo che non esiste un singolo digit esadecimale che rappresenti il nostro 17.
Allora, per formarne l'equivalente esadecimale, dobbiamo fare un altro passo, dividendo 17_{10} per 16_{10}:

16 | 2 8 2
16 | 1 7 r 10
1 r 1

$282_{10} = 11A_{16}$

Convertiamo ora gli stessi due numeri decimali, 124_{10} e 282_{10} al loro equivalente ottale. Poiché i numeri ottali sono in base otto, per generare l'equivalente ottale, dovremo dividere ripetutamente il numero decimale per 8_{10}, come segue:

8 | 1 2 4
8 | 1 5 r 4
1 r 7

$124_{10} = 174_8$

8 | 2 8 2
8 | 3 5 r 2
4 r 3

$282_{10} = 432_8$

Ora dobbiamo assicurarci che le nostre conversioni siano corrette controllando gli equivalenti ottale ed esadecimale di 124_{10} e 282_{10} per mezzo dei loro intermediari binari:

1	7	4
001	111	100

0	7	C
0	0111	1100

4	3	2
100	011	010

1	1	A
1	0001	1010

$124_{10} = 174_8 = 7C_{16}$ $\qquad$ $282_{10} = 432_8 = 11A_{16}$

Le conversioni ottale ed esadecimale sono realmente esatte. **Il convertitore numeri ottali ed esadecimali al loro equivalente decimale è abbastanza facile se ricordate cosa rappresentano i vari digit di un numero ottale od esadecimale.**
Per mantenere semplici le cose, consideriamo i numeri a quattro digit; la rappresentazione decimale di ciascun numero a quattro digit può essere definita dalla seguente equazione:

$$\text{Valore decimale} = P \times (\text{base})^3 + Q \times (\text{base})^2 + R \times (\text{base}) + S$$

Un numero con più di quattro digit semplicemente avrà alla sinistra dei termini con la base elevata a potenze più alte.
Ora l'equazione generale per i quattro digit può essere riscritta, per i casi specifici di numeri ottali ed esadecimali, come segue:

$$\text{Valore decimale} = P \times 8^3 + Q \times 8^2 + R \times 8 + S \text{ (ottali)}$$
$$\text{Valore esadecimale} = P \times 16^3 + Q \times 16^2 + R \times 16 + S \text{ (esadecimali)}$$

All'atto di convertire un numero ottale od esadecimale al suo equivalente decimale, moltiplicherete ciascun digit del numero ottale od esadecimale per l'appropriato moltiplicatore base.

Vediamo un paio di esempi:

$$
\begin{aligned}
2473_8 &= (2\text{x}8^3 + 4\text{x}8^2 + 7\text{x}8 + 3)_{10} \\
&= (2\text{x}512 + 4\text{x}64 + 7\text{x}8 + 3)_{10} \\
&= (1024 + 256 + 56 + 3)_{10} \\
&= 1339_{10}
\end{aligned}
$$

$$
\begin{aligned}
149A_{16} &= (1\text{x}16^3 + 4\text{x}16^2 + 9\text{x}16 + 10)_{10} \\
&= (1\text{x}4096 + 4\text{x}256 + 9\text{x}16 + 10)_{10} \\
&= (4096 + 1024 + 144 + 10) \\
&= 5274_{10}
\end{aligned}
$$

CODICI DI CARATTERE

I commutatori a due stati usati dalla logica dei microcomputer per generare digit binari devono essere anche usati per rappresentare lettere dell'alfabeto ed ogni carattere capace di essere visualizzato, stampato o manipolato in qualche altro modo.

Se, come abbiamo stabilito all'inizio di questo capitolo, la logica del computer consiste di null'altro che una serie di commutatori a due stati, allora non abbiamo alternative se non usare serie di commutatori (e quindi digit binari) anche per rappresentare caratteri.

Onde uscirne fuori con qualche ragionevole tecnica di codificare caratteri, dobbiamo esplorare due problemi:

1) **C'è qualche metodo "naturale per rappresentare caratteri, come c'è per rappresentare dati binari"?**
2) **Come distingueremo fra una serie di digit binari rappresentanti un carattere, confrontati con la serie di digit binari rappresentanti numeri o ogni altra informazione?**

Non esiste alcun metodo naturale di rappresentare caratteri usando codici a digit binari, e ogni codice a digit binari che possiamo generare potrebbe essere anche interpretato come un numero binario.

Ancora una volta incontriamo la necessità per voi, come programmatore, di conoscere in anticipo ciò che rappresenta una sequenza di digit binari numerici.

E ancora una volta potete contare con sicurezza che l'uso multiplo di digit binari non crea mai problemi.

Gli svariati codici usati per rappresentare caratteri usano tutti un byte (8 digit binari) per rappresentare un singolo carattere. Un byte ha 256 differenti combinazioni possibili di 8 digit binari.

numero di bit / un byte

7	6	5	4	3	2	1	0	valore decimale dal reticolo di bit
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	0	2
–	–	–	–	–	–	–	–	---
1	0	0	0	0	0	0	1	129
1	0	0	0	0	0	1	0	130
1	0	0	0	0	0	1	1	131
–	–	–	–	–	–	–	–	---
1	1	1	1	1	1	0	1	253
1	1	1	1	1	1	1	0	254
1	1	1	1	1	1	1	1	255

Cosicché un byte si può interpretare come:

1) un numero positivo con un valore decimale compreso fra 0 e + 255;
2) un numero con segno, con un valore decimale compreso fra –128 e + 127;
3) un byte di un numero multibyte con o senza segni;
4) un codice di carattere.

ASCII

Lo schema più popolare di codice usato per rappresentare caratteri è conosciuto come l'American Standard Code for Information Interchange, generalmente indicato come ASCII.

La serie completa del codice ASCII per tutti i caratteri stampabili sarà data in Appendice. Solo i codici di caratteri ASCII per i digit numerici da 0 a 9 hanno una base logica per la loro selezione.

Se voi guardate a questi codici di carattere vedrete che i quattro digit binari di ordine elevato eguagliano il valore numerico associato col loro carattere.

Codice binario	Equivalente esadecimale	Carattere ASCII
00110000	30	0
00110001	31	1
00110010	32	2
00110011	33	3
00110100	34	4
00110101	35	5
00110110	36	6
00110111	37	7
00111000	38	8
00111001	39	9

Da notare che in appendice riportiamo la serie dei digit binari rappresentanti ciascun codice di carattere usando il suo equivalente esadecimale; ciò rende i codici molto più facili da leggere. **Possiamo indicare l'equivalente numerico di una striscia di testo usando i digiti esadecimali come segue:**

```
T  h  i  s     i  s     t  h  e     n  u  m  e  r  i  c
54 68 69 73 20 69 73 20 74 68 65 20 6E 75 6D 65 72 69 63 0D

e  q  u  i  v  a  l  e  n  t     o  f     a
65 71 75 69 76 61 6C 65 6E 74 20 6F 66 20 61 0D

t  e  x  t     s  t  r  i  n  g
24 65 78 74 20 73 74 72 69 6E 67 2E
```

Ciascuna delle lettere di testo ha sotto di sé i due digit esadecimali che rappresentano il codice ASCII per le lettere, come definito in appendice. Da notare che fra le varie parole compare il codice 20_{16}; è il codice per la spaziatura. Il codice $0D_{16}$ rappresenta un ritorno carrello.

STRING

Il termine "string" (pressoché equivalente catena, o striscia, come qui sopra l'abbiamo definita) viene usato comunemente per descrivere una sequenza di caratteri immagazzinati mediante i loro codici numerici. Entro un computer il vostro testo sarà immagazzinato come una sequenza di bit.

Quando avete bisogno di stampare questo testo, voi andate a prendere i bit nella sequenza esatta e li trasmettete ad un display o ad una stampante.
La logica associata col display o con la stampante interpreta il dato binario assumendo che esso rappresenta dei caratteri.

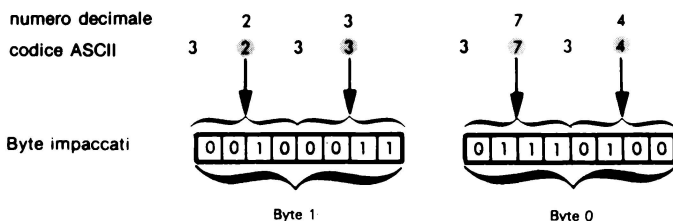
Se inserite del testo mediante una tastiera, allora, ogni volta che voi premete un tasto, il codice di digit binario associato con il tasto che poi abbassate viene trasmesso al microcomputer, che immagazzina il codice in un'appropriata memoria.

Voi potete modificare i codici trattandoli come dati binari. Supponiamo, per esempio, che abbiate una grossa quantità di dati numerici che desiderate immettere in memoria, e non abbiate alcun dato alfabetico. Se esaminate ancora la tabella ASCII in appendice, vedrete che tutti i digit decimali hanno gli stessi quattro bit di ordine elevato:

il codice ASCII per il digit decimale N è $3N_{16}$

BYTE IMPACCATI

Potete risparmiare un sacco di memoria "impaccando" i digit decimali, due per byte, come segue:

numero decimale	2	3	7	4
codice ASCII	3 2	3 3	3 7	3 4
Byte impaccati	0 0 1 0	0 0 1 1	0 1 1 1	0 1 0 0
	Byte 1		Byte 0	

Effettuando le operazioni illustrate in fig. 4-1 sui vostri codici di carattere, potete ricreare i caratteri ASCII.

LOGICA DEI COMPUTER E OPERAZIONI BOOLEANE

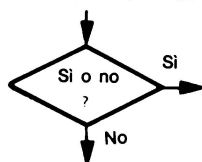
Un microcomputer spenderà ben poco del suo tempo facendo dell'aritmetica; esistono infatti molti programmi che non contengono alcunché di aritmetica. Un computer spenderà la maggior parte del suo tempo effettuando operazioni "logiche".

STATUS FLAG

Se esaminate il grafico di flusso per la logica di programma dato nel secondo capitolo, vedrete frequentemente il seguente tipo di passaggio decisionale.

Si o no ?

Sì

No

Un metodo comune per maneggiare semplice logica di decisione a due vie, come sopra illustrato, è di provvedere il microcomputer con alcuni speciali commutatori chiamati "status flag" (segnalatori di stato).

Gli eventi che precedono il passo logico devono porre uno di questi interruttori "off" o "on". La logica che produce la decisione a due vie diventa allora una singola istruzione che può essere illustrata come segue:

se il flag è on, devia sull'istruzione X
se il flag è off, continua con la prossima istruzione

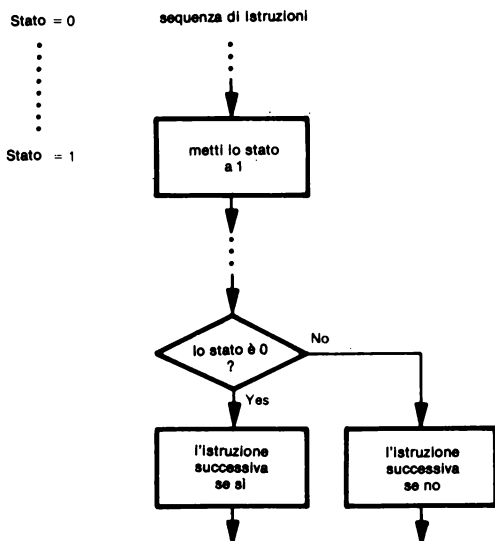
Gli status flag rappresentano una delle forme più semplici della logica da micro-

Esempio di dati (usando il byte 1)	Logica di programma
00100011	preleva un bit impaccato
00100000	isola quattro bit di ordine elevato (o nibble)
00000010	sposta 4 bit di ordine elevato su 4 bit di ordine basso
00110010	aggiungi 30_{16}
	da uscita alla stampante o al display
00100011	preleva ancora lo stesso byte impaccato
00000011	isola quattro bit di basso ordine (o nibble)
00110011	aggiungi 30_{16}
	da uscita alla stampante o al display

Figura 4-1. Disassemblaggio di byte impaccato e logica di formazione del codice ASCII

computer. Microcomputer diversi hanno numeri e tipi differenti di status flag, ma non è necessario identificarli a questo punto.

Per capire il concetto di uno status flag, tutto quello cui dovete pensare è un'interruttore a due vie che le istruzioni nel vostro programma possono commutare "on" o "off"; le successive istruzioni controllano l'interruttore per determinare quale delle due strade prenderà la vostra logica di programma.

Stato = 0
sequenza di istruzioni
Stato = 1
metti lo stato a 1
lo stato è 0 ?
No
Yes
l'istruzione successiva se si
l'istruzione successiva se no

OPERATORI LOGICI

OPERATORI LOGICI

La maggior parte della logica dei computer è prodotta da quattro "operatori logici". Non c'è nulla di particolarmente misterioso a proposito degli operatori logici; l'addizione, la sottrazione, la moltiplicazione e la divisione sono "operatori aritmetici".

Un operatore logico prende l'ingresso dati, opera qualcosa di non aritmetico su di esso, e crea un risultato, esattamente come un operatore aritmetico prende l'ingresso dati, vi opera qualcosa di aritmetico e genera un risultato.

I quattro operatori logici sono: NOT, AND, OR ed OR esclusivo, vediamoli uno alla volta.

L'OPERATORE NOT

L'operatore NOT è il più semplice da capire; esso semplicemente dice: muovere un interruttore nella sua posizione contraria. Ciò vuol dire che, se è in posizione "on" va messo in "off", o viceversa. Esaminando l'effetto di un'operatore NOT

su di un bit, esso converte un 1 in uno 0, uno 0 in un 1; ciò si può illustrare come segue:

NOT 0 = 1
NOT 1 = 0
NOT 101101 = 010010

Più frequentemente si usa una barra sopra il numero anziché l'indicazione NOT; si scrive quindi

$$\overline{0} = 1$$
$$\overline{1} = 0$$
$$\overline{101101} = 010010$$

COMPLEMENTO UNO

L'operatore NOT genera il complemento uno di un numero; ricordate che il primo passo per creare i complementi due di un numero è crearne i complementi uno.

L'OPERATORE AND

L'operatore AND controlla che due interruttori siano contemporaneamente "on". Le operazioni AND si possono così indicare:

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

In genere si usa un punto anziché la notazione AND; quindi le quattro operazioni qui sopra indicate possono essere riscritte:

$0 \cdot 0 = 0$
$0 \cdot 1 = 0$
$1 \cdot 0 = 0$
$1 \cdot 1 = 1$

La logica dell'operazione AND è parte comune della nostra vita di ogni giorno. Un classico esempio è quello dei due fratellini che vogliono il gelato e ne possono comprare uno solo.

Se Gianni vuole il gelato A
e Paolo vuole il gelato B
non si compra nessun gelato: $A \cdot B = 0$

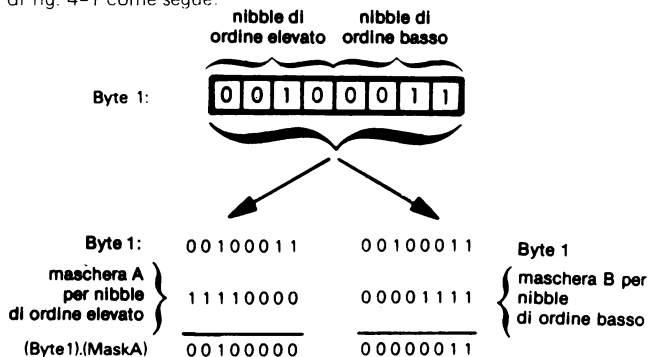
Se Gianni vuole il gelato A
se anche Paolo vuole il gelato A,
si può comprare il gelato A: $A \cdot A = 1$

MASCHERA DI BIT

Abbiamo visto anche un'applicazione dell'operazione AND. Richiamiamo ora quanto spiegato sul come i codici di carattere per digit numerici possono essere immagazzinati due per byte. In fig. 4-1 si potrebbe usare una maschera ad 8 bit ed un'operazione AND per usare uno o l'altro nibble numerico; si può illustrare questo per il byte 1 di fig. 4-1 come segue:

nibble di ordine elevato **nibble di ordine basso**

Byte 1: 0 0 1 0 0 0 1 1

Byte 1:	00100011	00100011	**Byte 1**
maschera A per nibble di ordine elevato	11110000	00001111	**maschera B per nibble di ordine basso**
(Byte1).(MaskA)	00100000	00000011	

Ovunque un bit 0 debba essere inserito in una sequenza di bit di dati, noi provvediamo ad inserire un bit 0 in una maschera AND. Ovunque la maschera AND abbia un bit 1, essa viene attraversata dai bit di dati inalterata.

Supponete, per esempio, di dover conservare inalterati i bit 2, 3, 4 e 5 di un byte, ma di aver bisogno che i bit 0 e 1, e 6 e 7 siano 0. Vi serve un AND sui dati binari con la maschera 00111100, che si può così illustrare:

	7	6	5	4	3	2	1	0 ← **numero di bit**
dati binari:	X	X	X	X	X	X	X	X
maschera:	0	0	1	1	1	1	0	0
maschera dati AND:	0	0	X	X	X	X	0	0

L'OPERATORE OR

L'operazione logica OR è un controllo su OGNI interruttore "on". L'operazione OR si può quindi così definire:

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

Si usa in genere il segno + invece della notazione OR; possiamo quindi riscrivere le quattro operazioni come segue:

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 1$$

Il fatto che si possa usare un segno più per rappresentare l'operatore logico OR può portare a confondervi, ma è irrilevante per un computer.
Ci saranno infatti istruzioni distinte, coi loro singoli codici indipendenti di istruzione, che rappresentano un'operazione di addizione o un'operazione logica OR.

Comunque il segno + usato per rappresentare sia una somma che un'operazione OR si verificherà solo in materiale scritto o stampato.

L'operazione OR ci è ancora una volta familiare nella nostra vita di tutti i giorni. Per esempio, nel caso precedente dei due fratellini, si acquisterà il biscotto x se uno o l'altro lo vogliono:

vuole Gianni il biscotto x?		vuole Paolo il biscotto x?		si compra il biscotto x?
no (= 0)	+	no (= 0)	=	no (= 0)
no (= 0)	+	sì (= 1)	=	sì (= 1)
sì (= 1)	+	no (= 0)	=	sì (= 1)
sì (= 1)	+	sì (= 1)	=	sì (= 1)

Possiamo usare l'operatore OR sul nostro esempio di carattere a digit decimale (fig. 4-1), onde fornire i quattro bit di ordine elevato per il codice di carattere ASCII, come segue:

	7	6	5	4	3	2	1	0 ← Bit No.
nibble isolato:	0	0	0	0	0	0	1	1
maschera OR per i quattro bit	0	0	1	1	0	0	0	0
di ordine elevato:	0	0	1	1	0	0	1	1

Ovùnque si debba inserire un bit 1 in una sequenza di bit di dati, noi forniamo un bit 1 in una maschera OR. Ovunque la maschera OR abbia un bit 0, essa lascia passare i bit inalterati, come segue:

	7	6	5	4	3	2	1	0 ← Bit No.
dati binari:	X	X	X	X	X	X	X	X
maschera arbitraria:	0	0	1	1	1	1	0	0
maschera dati OR:	X	X	1	1	1	1	X	X

Cosicché, l'operatore AND può essere usata come maschera di "azzeramento", mentre l'operatore OR può essere usato come maschera di "inserimento".

L'OPERATORE XOR

L'ultimo operatore logico che descriviamo è l'OR esclusivo, o XOR. L'OR esclusivo controlla differenze e cambiamenti, come segue:

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0

Anziché la notazione XOR si usa in genere il simbolo "più" cerchiato; riscriviamo quindi le quattro operazioni:

$$0 \oplus 0 = 0$$
$$0 \oplus 1 = 1$$
$$1 \oplus 0 = 1$$
$$1 \oplus 1 = 0$$

L'OR esclusivo è esso pure parte della nostra logica di vita quotidiana; per esempio

scoppia una zuffa fra i nostri due fratellini ogni volta che Gianni dice sì Paolo dice no, o viceversa:

opinione di Gianni		opinione di Paolo		la zuffa?
no (= 0)	⊕	no (= 0)	=	no (= 0)
no (= 0)	⊕	sì (= 1)	=	sì (= 1)
sì (= 1)	⊕	no (= 0)	=	sì (= 1)
sì (= 1)	⊕	sì (= 1)	=	no (= 0)

Nella logica da computer si userà l'OR esclusivo per controllare cambiamenti di stato. Supponiamo, per esempio, che il sapere che un interruttore è "on" o "off" sia insufficiente, e vi serva anche conoscere se l'interruttore è stato commutato "on" o "off" dopo l'ultima volta che l'avete controllato.

Potete allora memorizzare la condizione di ciascun interruttore ogni volta che lo provate quindi paragonarne la posizione con la condizione memorizzata come segue:

interruttore		ultima posizione	
X	⟶	X	memorizzate la posizione
Y			nuova posizione
Y	⊕	X	comparate le posizioni e controllate i cambiamenti

Così, eseguendo un'operazione XOR sulla condizione dell'interruttore e sulla sua precedente posizione, potretè individuare se l'interruttore ha cambiato posizione dall'ultimo controllo.

Capitolo 5
DENTRO IL COMPUTER

Abbiamo finora descritto, nei primi tre capitoli, i concetti fondamentali di un microcomputer; quindi, nel quarto capitolo, siamo passati all'altro estremo, definendo i concetti basilari grazie ai quali si può creare qualsiasi funzione del computer.

Ora è tempo di colmare il fosso che esiste fra i concetti fondamentali ed il prodotto finale; il sistema microcomputer; è ciò che faremo nei prossimi due capitoli.

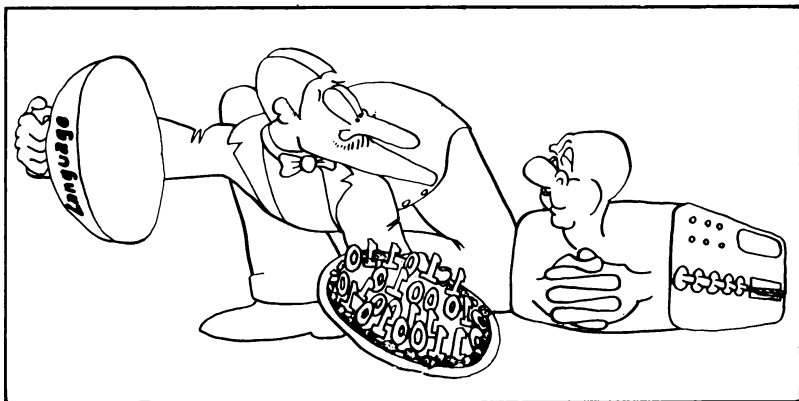
In questo primo, esamineremo il microcomputer stesso, separandone e studiandone i vari componenti; nel successivo, esamineremo il modo in cui i concetti logici digitali di base possono essere usati per creare i componenti del sistema microcomputer qui descritto.

A PROPOSITO DEI LINGUAGGI DI PROGRAMMA

Esistono dei cosiddetti linguaggi a "più alto livello", come il BASIC, il FORTRAN ed il COBOL; ci sono anche i linguaggi "fondamentali" di programmazione, indicati come "linguaggio di assembly".

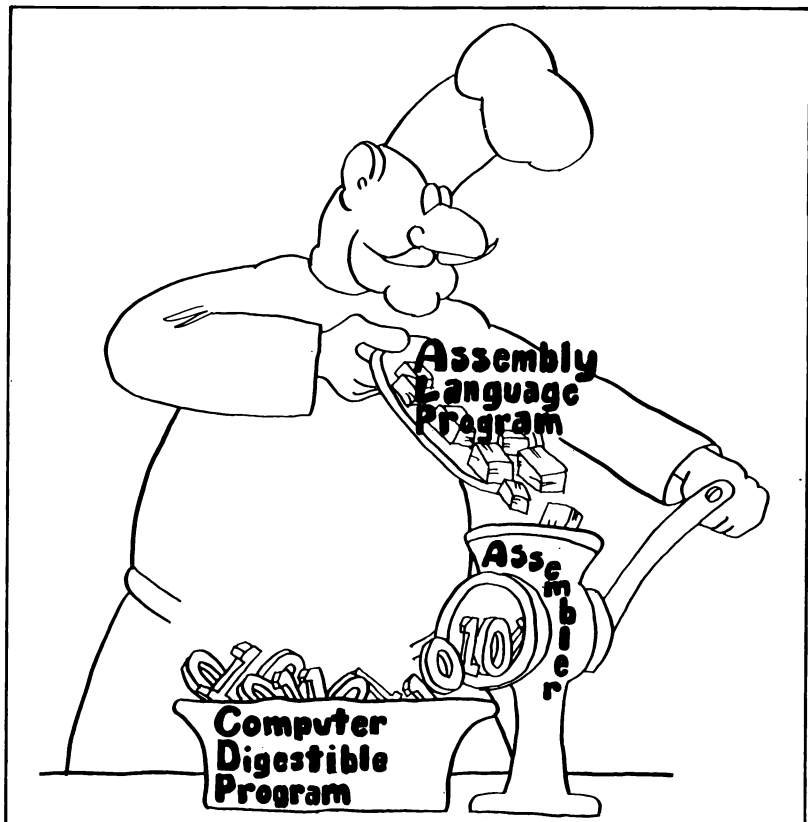
Riferendoci ad ogni linguaggio di programma, il punto più importante da capire è che un **linguaggio di programma è una comodità per il programmatore.**

Un linguaggio di programma è una creazione artificiale, progettata per rendere più facile la vostra vita di programmatore.

Qualunque sia il linguaggio che voi deciderete essere il migliore, **il computer richiede sempre di ricevere il programma come una sequenza di numeri:**

Ora sarà lo stesso computer a prendersi cura di convertire il programma dalla forma in cui voi, il programmatore, lo scrivete, alla forma in cui esso, il computer, lo può capire ed eseguire.

Allo scopo di effettuare questa conversione, il computer esegue un altro programma, un programma che qualcun altro ha scritto per voi.

ASSEMBLER

Un programma chiamato **"assembler" converte i programmi che voi scrivete, in linguaggio assembly, in programmi che il computer può capire ed eseguire.**

COMPILATORE

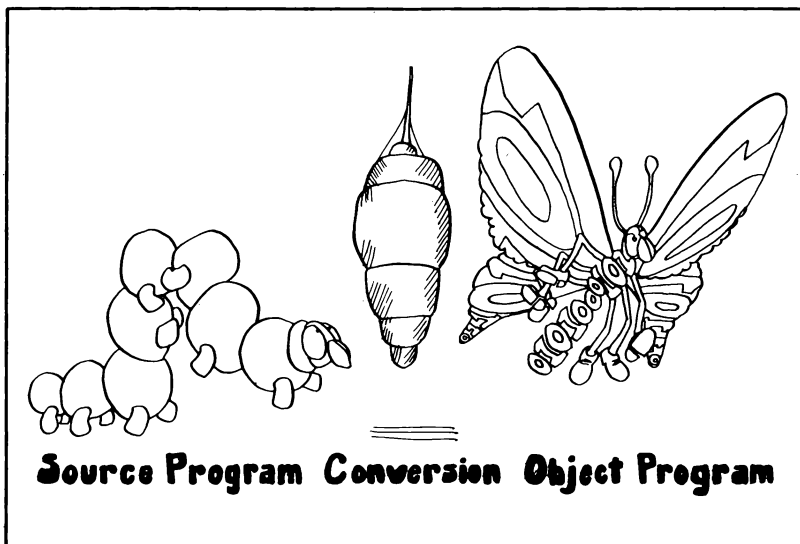
Un programma identificato come "compilatore" adempie allo stesso compito di conversione per i programmi che voi scrivete usando un linguaggio a più alto livello.

Assembler e compilatori trattano il vostro programma come dati; essi lo leggono sotto forma di dati e lo convertono in un'altra forma di dati (la versione del vostro programma eseguibile dal computer).

SOURCE PROGRAM
PROGRAMMA OGGETTO

Indichiamo un programma in forma umanamente comprensibile come un "source program" (programma sorgente). Vale a dire che un source program è un programma scritto in un qualche linguaggio di programmazione.

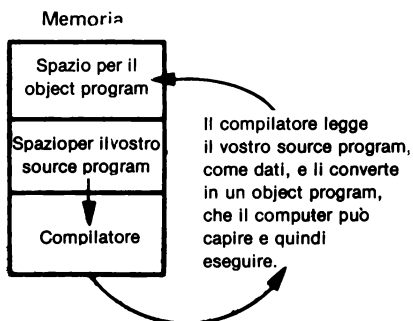
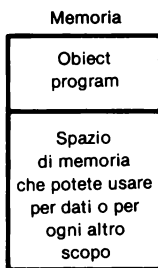
Una volta che il programma è stato convertito nella sua forma comprensibile al computer, esso è chiamato "object program" (programma oggetto).

Un object programma non è altro che una sequenza di numeri. Tutto ciò può essere illustrato come segue:

Così assembler e compilatori leggono in una certa forma di dati (il vostro source program) e li convertono in un'altra forma di dati (un object program).

In realtà vi sono due tipi di compiler. Un tipo prende il vostro programma, lo converte in forma comprensibile al computer e lo conserva. Susseguentemente, la forma comprensibile al computer viene caricata in memoria per la esecuzione. Ciò si può illustrare come segue:

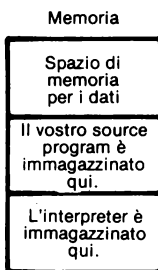
1° passo - passo di compilazione

2° passo - passo di esecuzione

Memoria
Obiect program
Spazio di memoria che potete usare per dati o per ogni altro scopo

INTERPRETER

Un altro tipo di compilatore non conserva mai la forma comprensibile al computer (per esempio, l'object program). Questo tipo di compilatore è chiamato "interpreter" (interprete).

Quando usate un interpreter, tutto il vostro source program risiede in memoria, assieme con l'interpreter, per tutto il tempo nel quale si sta eseguendo il source program. Questo si può illustrare come segue:

Memoria
Spazio di memoria per i dati
Il vostro source program è immagazzinato qui.
L'interpreter è immagazzinato qui.

L'interpreter converte il vostro source program nel codice object come richiesto, e come qui illustrato.

Memoria

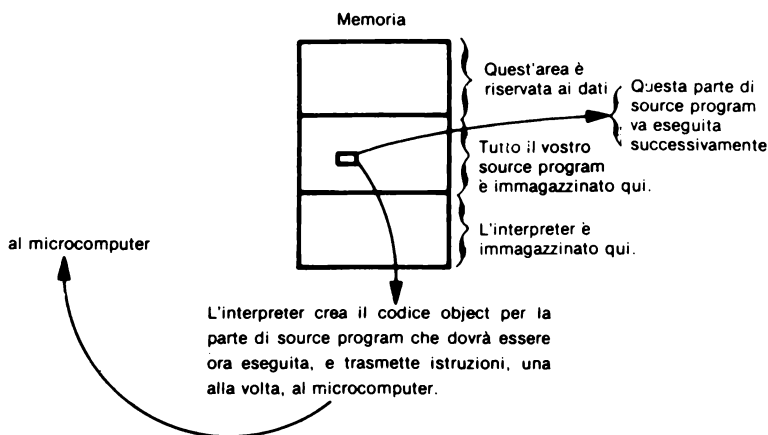
Quest'area è riservata ai dati

Questa parte di source program va eseguita successivamente

Tutto il vostro source program è immagazzinato qui.

L'interpreter è immagazzinato qui.

al microcomputer

L'interpreter crea il codice object per la parte di source program che dovrà essere ora eseguita, e trasmette istruzioni, una alla volta, al microcomputer.

La figura mostra un'area di memoria che viene messa da parte per il vostro intero source program. Potreste essere indotti in errore pensando che l'ammontare di memoria messo da parte per il vostro source program pone un limite superiore alla dimensione del source program eseguibile.

In effetti, voi potete eseguire programmi molto più ampi almeno fintanto che il programma più grande può essere spezzato in blocchi, nessuno dei quali sovrasti lo spazio di memoria disponibile per il source program.

Gli stessi compilatori e interpreter sono object program che qualcuno altro ha scritto per Voi.

Possiamo spiegare la differenza fra un compilatore ed un interpreter in termini non tecnici pensando ai modi in cui un attore può imparare a recitare le battute di un soggetto in uno spettacolo.

Pensate al source program come al manoscritto dell'attore; le istruzioni dell'object program che vanno al microcomputer equivalgono all'attore che sta recitando le sue battute ad una recita.

Se l'attore impara tutta la sua parte, quindi getta via il testo e recita le sue battute, quello che ha fatto è equivalente al compilare un source program.

Ma supponiamo che l'attore non impari la sua parte tutta intera; supponiamo che l'attore conservi il suo testo ed abbia un suggeritore che gli mostri le sue battute una alla volta, usando una tavola visualizzatrice.

Ora egli recita le sue battute alla stregua di un interpreter. Il BASIC è il più popolare linguaggio da microcomputer ad alto livello; esso è anche un linguaggio interpreter.

In conclusione, possiamo dividere la maggior parte dei linguaggi di programmazione in linguaggi di "più alto livello" e in linguaggi di "assembly". I linguaggi di alto livello sono convertiti in object program dai compilatori ed interpreter. I linguaggi assembly sono convertiti in codice object da un assembler.

La differenza principale fra linguaggi ad alto livello e linguaggi assembly sta nel fatto che i linguaggi ad alto livello sono progettati per rappresentare problemi, mentre i linguaggi assembly sono progettati per rappresentare il computer.

Cosicché un computer vede un source program a linguaggio di alto livello come una cosa molto estranea, ed un compilatore ha il suo bel da fare a convertire il source program in un object program.

Al contrario, un source program in linguaggio assembly può essere convertito in un object program con tutta facilità; un assembler risulta perciò essere un programma relativamente semplice.

Passiamo ora a confrontare i linguaggi ad alto livello ed i linguaggi assembly allo scopo di identificare più chiaramente le differenze fra i due.

CONFRONTO FRA LINGUAGGI AD ALTO LIVELLO E LINGUAGGI ASSEMBLY

Esaminiamo prima di tutto i vantaggi dei linguaggi ad alto livello. **I linguaggi ad alto livello sono più facili ad usare dei linguaggi assembly;** ciò in quanto i linguaggi ad alto livello rappresentano il problema piuttosto che il computer.

Per esempio, una semplice addizione, usando un linguaggio ad alto livello, potrebbe essere scritta in questa forma auto-illustrante:

SUM = VAL 1 + VAL 2

VAL 1 e VAL 2 sono nomi che potete assegnare a due addendi, che possono avere qualsiasi valore; SUM è evidentemente il nome che assegnate alla somma dei due.

Il linguaggio assembly si presenta a voi con una manifestazione del microcomputer, cioè in forma umanamente comprensibile. Quindi l'addizione qui sopra illustrata verrà programmata, in linguaggio assembly, come segue:

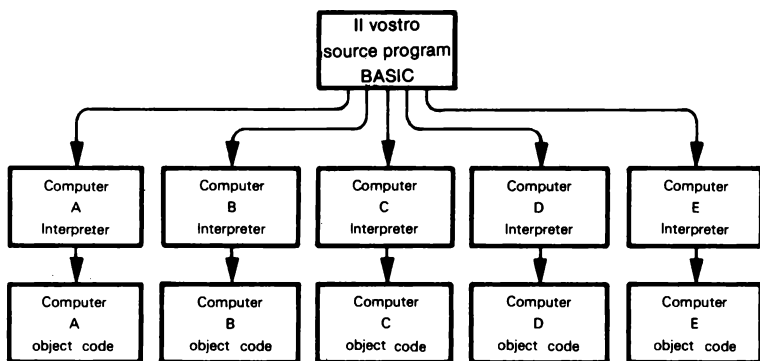
```
LXI     H,VAL 1
LDA     VAL 2
ADD     A,M
STA     SUM
```

VAL 1 e VAL 2 non sono più nomi da voi assegnati agli addendi; VAL 1 e VAL 2 sono ora indirizzi; essi identificano locazioni di memoria in cui vengono immagazzinati gli addendi. Quindi gli addendi devono essere, ciascuno, abbastanza piccoli da entrare in una locazione di memoria.
Analogamente, SUM è l'indirizzo della locazione di memoria in cui verrà immagazzinata la somma, a patto che essa riesca a stare in una word di memoria.

La definizione di un'addizione in linguaggio assembly non è più autospiegante. C'è un altro importante vantaggio associato col fatto che i linguaggi ad alto livello sono "orientati secondo problema" (intendiamo, con questo termine, significare che il linguaggio non è progettato con alcun computer in mente).

Allora, se voi scrivete un programma in un linguaggio ad alto livello, potete convertire questo source program in un object program che potrà funzionare su qualsiasi computer, a patto che esso abbia un compilator (o un interpreter) per il vostro linguaggio ad alto livello.

Supponiamo per esempio che voi scriviate un programma in BASIC; voi potete eseguire questo programma sul vostro computer; ed anche i vostri amici e conoscenti potranno eseguire il vostro programma sui loro computer completamente diversi, ad unica condizione che anche tali computer abbiano gli interpreter BASIC.

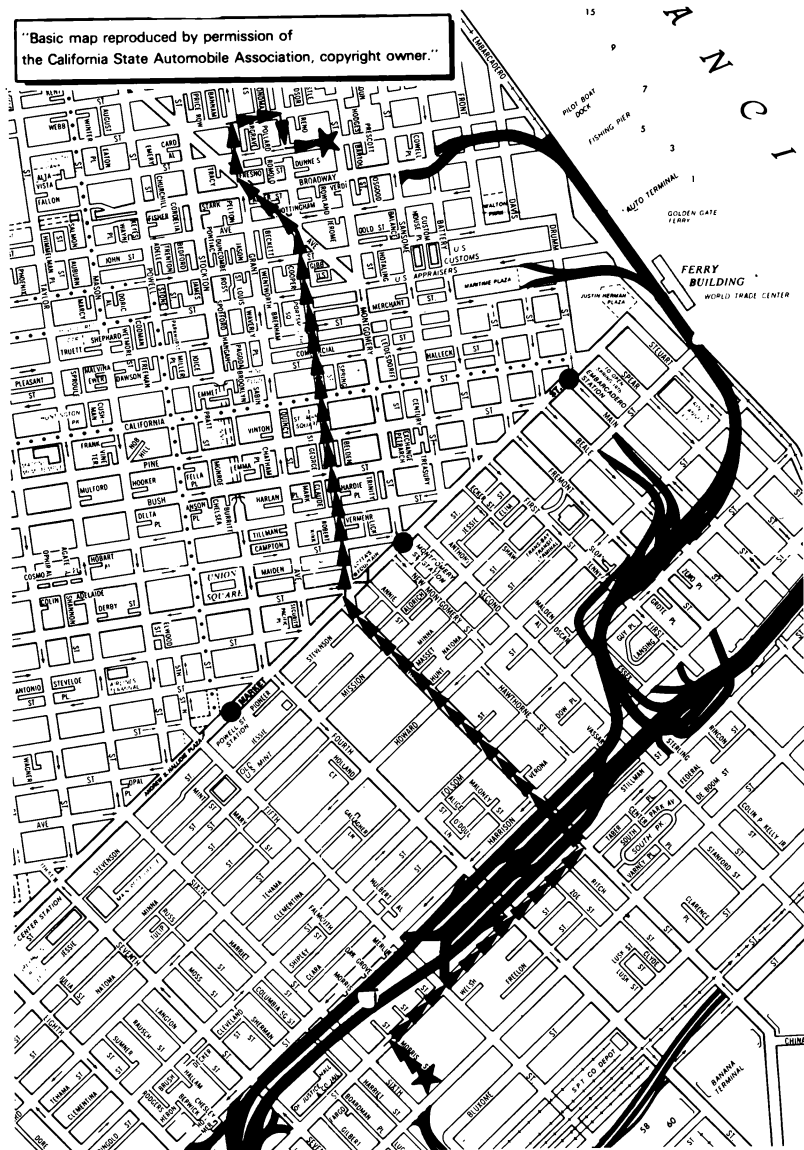
Tutto ciò può essere inquadrato nella seguente rappresentazione:

Il linguaggio assembly, d'altra parte, è una rappresentazione umana del computer che state usando. Quindi ogni singolo computer e microprocessore ha il suo personale, unico linguaggio assembly; ed un programma scritto nel linguaggio assembly di un computer o microprocessore è totalmente incomprensibile a qualsiasi altro microcomputer o microprocessore.

Se voi scrivete un source program in tale linguaggio per il vostro microprocessore, solo coloro che possiedono microcomputer equipaggiati col vostro microprocessore potranno assemblare e far lavorare il vostro source program.

In teoria, sarebbe possibile scrivere un programma simile ad un compilatore che prendesse un source program scritto nel linguaggio assembly di un microprocessore

e lo convertisse in un object program per un altro microprocessore.

In realtà, pochi lo fanno, in quanto il linguaggio assembly di un altro microprocessore è altrettanto estraneo e difficile da trattare quanto un linguaggio ad alto livello.

Con tutti i vantaggi che derivano dal programmare in linguaggio elevato, perché mai qualcuno dovrebbe cimentarsi col linguaggio assembly? Tuttavia quest'ultimo linguaggio ha dei vantaggi.

In primo luogo, **il linguaggio assembly dà origine ad object program molto più brevi dei linguaggi ad alto livello.**

Ciò avviene in quanto il linguaggio assembly per ciascun microprocessore o computer, è specificamente realizzato per quel microprocessore o computer.

In effetti, un object program creato da un compilatore da un source program a linguaggio elevato è usualmente lungo da 2 a 4 volte lo stesso object program creato da un assembler da un source program in linguaggio assembly.

Ciò avviene in quanto il compilatore deve, in realtà, scrivere un programma in linguaggio assembly per presentare il problema, quale definito nel linguaggio ad alto livello. Ma, mentre un programmatore può scrivere un programma in linguaggio assembly usando il discernimento umano, il compilatore deve fare il lavoro basandosi su regole fisse.

Considerate un'analogia della vita di tutti i giorni: dovete dare a qualcuno le indicazioni per andare da un punto ad un altro di una città.

Se voi conoscete esattamente il punto di partenza e la destinazione, oltre naturalmente alla città, sarete in grado di definire esattamente il percorso.
Cerchiamo ora di creare una serie di istruzioni di uso generale che voi possiate mettere assieme allo scopo di definire il percorso da seguire fra due punti qualsiasi di una qualsiasi città.

Se queste istruzioni debbono essere interpretate da una macchina, esse non possono lasciar nulla all'immaginazione; dovranno quindi esservi alcuni numeri fissi di istruzioni, quali:

gira a sinistra
gira a destra
prova per una strada a corsia unica
prova per una deviazione a 45°
ecc.

Non potrete includere istruzioni che presuppongano che voi sappiate se una strada è o no a senso unico, perché tali sensi unici sono soggetti a cambiare.

Analogamente, non potrete inserire istruzioni che semplicemente definiscano il numero di isolati da percorrere in linea retta, perché vi possono essere blocchi alla circolazione che impediscono questo percorso.
Una volta che voi partiate per definire una serie di regole direzionali di uso generale che tengano conto fattori contingenti indefinibili, potrete avere una qualche idea del problema affrontato da un compilatore.

Il compilatore non conosce quali possano essere le peculiarità di ciascun specifico computer, perciò esso deve produrre programmi che tengano in conto le possibilità più strane.
I linguaggi ad alto livello hanno un altro problema. Il compilatore che converte un source program a linguaggio elevato in un object program è da se stesso un grosso programma; un programma compilatore può essere lungo otto volte un programma assembler.

Cosicché, se il vostro sistema microcomputer non è sufficientemente ampio, voi non potrete usare un linguaggio elevato, in quanto il vostro sistema microcomputer non avrà memoria sufficiente per contenere il compilatore.
Se disponete di un interpreter, esso allora dovrà essere sempre in memoria, assieme al programma che state eseguendo. Ma questa differenza fra un compilatore ed un interpreter è già stata precedentemente illustrata in questo capitolo.

Il fatto che i source program a linguaggio elevato generino object program più lunghi significa anche che questi ultimi impiegheranno più tempo ad essere eseguiti, in quanto vi sono più istruzioni da seguire.

Se la vostra applicazione si scontra con problemi di tempo, potete accelerare le cose di almeno un paio di volte semplicemente riscrivendo il vostro programma in linguaggio assembly.

C'è anche da dire che alcuni dei vantaggi associati coi linguaggi ad alto livello non sono poi tutto ciò che sembrano essere.

Per esempio, si **presuppone che i linguaggi elevati siano "trasportabili",** cioè che un source program possa essere compilato ed eseguito da molti microprocessori differenti; ma questo non sempre è vero.

Spesso troverete che esistono differenze di poco conto nel modo in cui il compilatore di un computer attende che appaia il source program.

Comunque, anche nel peggiore dei casi, le modifiche che avrete da fare ad un source program a linguaggio elevato passando ad un nuovo microprocessore o computer, sono modeste se paragonate ai problemi connessi col riscrivere completamente il programma nel linguaggio assembly del nuovo microprocessore o computer.

Qual'è allora la conclusione? Se vi avviate ad usare un microcomputer semplicemente come mezzo per eseguire programmi, dovrete orientarvi verso linguaggi elevati il più presto possibile.

Se invece, prevedete di lavorare attorno al vostro microcomputer, costruendovene almeno una parte, modificandolo, estendendolo, o comunque trafficando con i suoi componenti, allora dovreste imparare il linguaggio assembly il più presto possibile, e con esso probabilmente rimarrete a lungo.

LOGICA FUNZIONALE DI UN MICROCOMPUTER

L'object program che voi create determina le funzioni che saranno eseguite dalla logica del vostro microcomputer.

La fig. 5-1 rappresenta, da un punto di vista funzionale, la logica di un microcomputer; si tratta della logica che ora passiamo ad esaminare.

Indipendentemente da ciò che il microcomputer si appresta a fare, in ultima analisi il suo compito si manifesta in questi tre passaggi:

1) portando i dati entro il microcomputer
2) modificando i dati
3) ritrasmettendo i dati modificati fuori dal microcomputer.

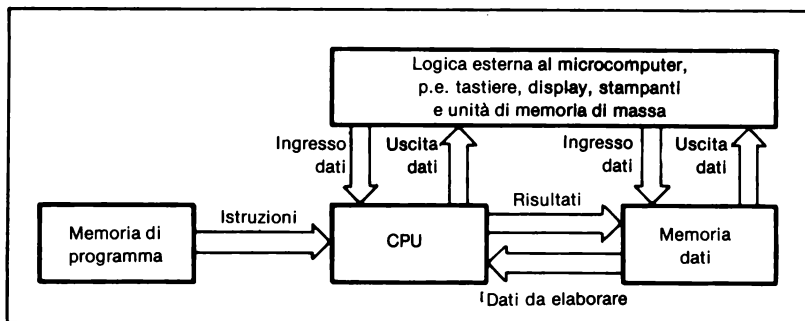



Figura 5-1 — La logica funzionale di un microcomputer

La logica al di fuori del microcomputer (che consiste di unità fisiche già descritte nei primi capitoli) viene usata per immettere informazioni, ricevere risultati ed immagazzinare grosse quantità di dati.

Il complesso di dati che è nel processo che sta per essere operato viene immagazzinato nella memoria dati, che (come potete ricordare dal 2° capitolo) è una memoria leggi/scrivi ad accesso rapido.

Perciò, i passaggi 1) e 3) di cui sopra vengono manipolati dalla logica del microcomputer ombreggiata in fig. 5-2.

Le unità fisiche, come ricorderete, trasmettono informazioni al e dal microcomputer, attraverso appropriate logiche di interfaccia. In riferimento alla fig. 5-1, ciò si può illustrare come segue:

Unità fisiche

Elettronica d'interfaccia

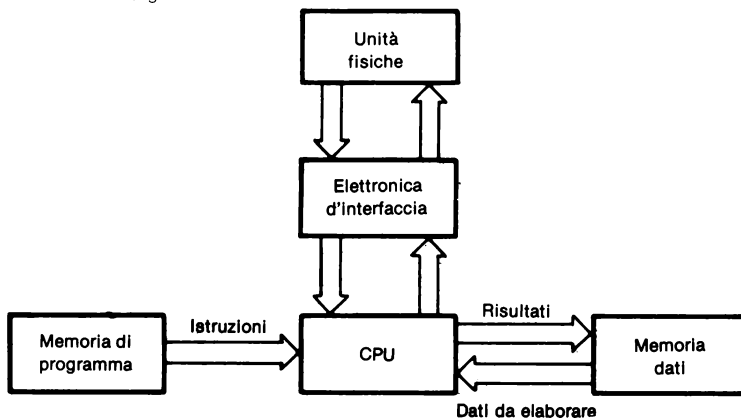
Memoria di programma

Istruzioni

CPU

Risultati

Memoria dati

Dati da elaborare

UNITA' DI ELABORAZIONE CENTRALE

Le operazioni che vengono attualmente effettuate sui dati sono eseguite da una logica entro l'Unità Centrale di Elaborazione (CPU). Queste operazioni sono definite da una sequenza di istruzioni che, prese assieme, costituiscono un programma; il programma viene immagazzinato nella memoria di programma.

Quindi il passo 2) dei tre indicati all'inizio del paragrafo è manipolato dalla logica ombreggiata in fig. 5-3.

MEMORIA DI PROGRAMMA

La memoria di programma può essere una memoria a sola lettura, (read only) o può essere una memoria legge/scrive (read/write). La memoria di programma può essere del tipo a sola lettura perchè le istruzioni sono trasmesse, dal programma immagazzinato in essa, alla CPU; ma le istruzioni non vengono usualmente trasmesse dalla CPU alla memoria di programma. Non è detto che la memoria di programma debba essere del tipo read only.

Nei sistemi a microcomputer è pratica comune separare i programmi dai dati, come indicato in fig. 5-1; inoltre, in molte applicazioni industriali, i programmi sono tenuti in memoria a sola lettura per essere sicuri che il programma non venga mai modificato o perduto accidentalmente.

Ma memoria di programma e memoria dati potrebbero essere una ed una sola; inoltre, è possibile, per una parte di un programma, trattare un'altra parte dello stesso come dati, nel quale caso il programma si cambia da solo.

Come potete aspettarvi, i programmi che si cambiano da soli possono diventare

molto complessi; cosicché, almeno finché siete alle prime armi, è prudente ragionare di memoria di programma e memoria dati come entità separate e distinte.

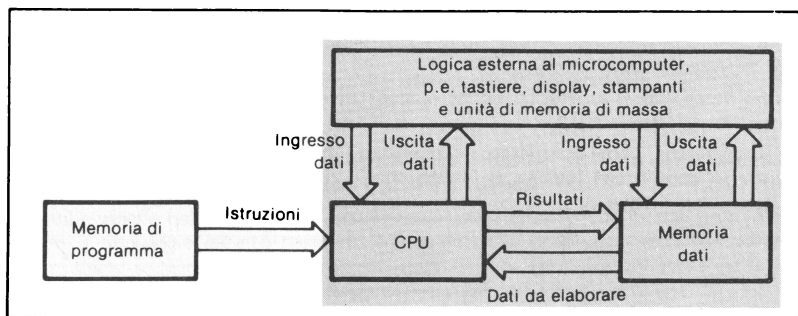



Figura 5-2. La logica funzionale di un microcomputer interessata dal movimento e immagazzinamento dati.

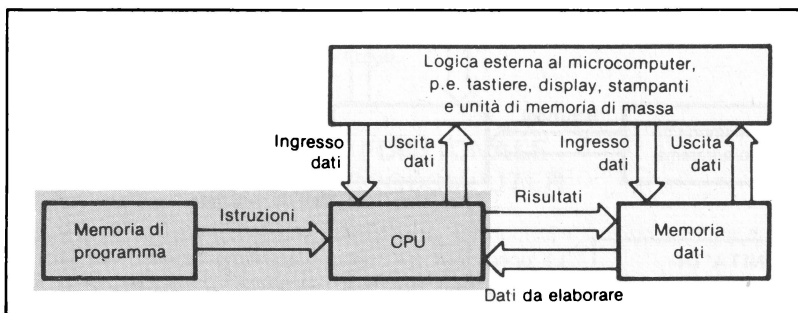



Figura 5-3. La logica funzionale di un microcomputer interessata alla modifica dati.

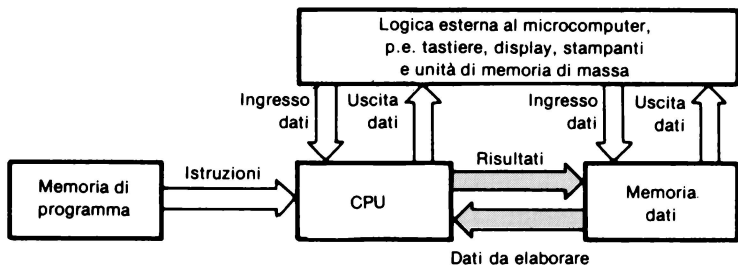
Il fatto che non abbiate ancora una buona comprensibilità di come lavorano le memorie di programma e di dati non è importante.

Nei primi capitoli abbiamo mostrato fotografie di chip contenenti tali memorie, e questi chip sono in grado di immagazzinare informazioni in forma comprensibile al computer; per ora questo è tutto quanto vi serve concretamente sapere sulla memoria di programma e di dati.

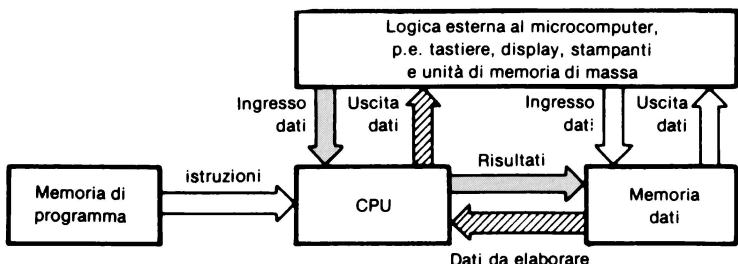
PERCORSI D'INFORMAZIONE

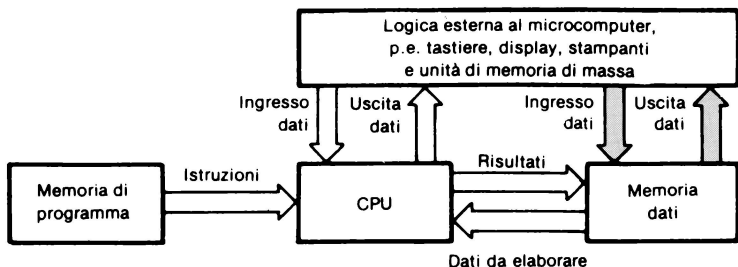
Esaminiamo ora i vari percorsi di informazione mostrati in fig. 5-1. Quando la CPU modifica i dati, essa normalmente attinge i dati da modificare dalla memoria dati,

e normalmente riporta i risultati alla stessa. **Ci sono perciò percorsi in ambedue le direzioni fra memoria dati e CPU.**

Logica esterna al microcomputer, p.e. tastiere, display, stampanti e unità di memoria di massa
Ingresso dati
Uscita dati
Ingresso dati
Uscita dati
Memoria di programma
Istruzioni
CPU
Risultati
Memoria dati
Dati da elaborare

I nuovi dati che entrano nel microcomputer viaggiano dalle unità fisiche esterne alla memoria dati attraverso la CPU.

I risultati in uscita viaggiano dalla memoria alle unità fisiche esterne sempre attraverso la CPU.

Logica esterna al microcomputer, p.e. tastiere, display, stampanti e unità di memoria di massa
Ingresso dati
Uscita dati
Ingresso dati
Uscita dati
Memoria di programma
istruzioni
CPU
Risultati
Memoria dati
Dati da elaborare

Il trasferimento di informazioni ad alta velocità fra floppy disk e memoria dati spesso avviene direttamente fra questi due dispositivi, saltando quindi la CPU:

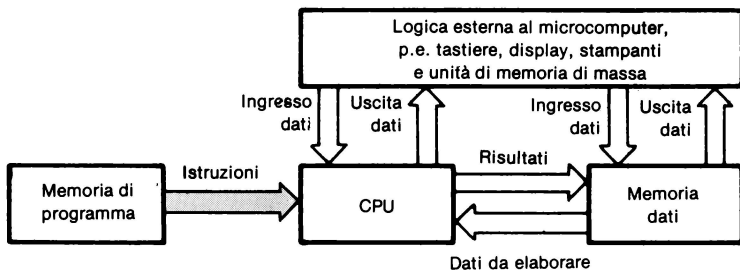
Logica esterna al microcomputer, p.e. tastiere, display, stampanti e unità di memoria di massa
Ingresso dati
Uscita dati
Ingresso dati
Uscita dati
Memoria di programma
Istruzioni
CPU
Risultati
Memoria dati
Dati da elaborare

DMA: MEMORIA AD ACCESSO DIRETTO

Il percorso dati qui sopra illustrato prende il nome di memoria ad accesso diretto, la cui terminologia inglese è abbreviata in DMA. Mentre la memoria deve essere ad un estremo del trasferimento dati a DMA, non è necessario che all'altro estremo ci sia un floppy disk, anche se ciò avviene di frequente. Qualsiasi logica esterna può costituire l'altro estremo del trasferimento dati a DMA.

Ogniqualvolta la CPU opera qualcosa (che stia muovendo o modificando dati, non

importa) una corrente di istruzioni trasmesse dalla memoria di programma della CPU controlla le operazioni della CPU stessa.

Deve perciò esservi un percorso unidirezionale per le informazioni che scorrono dalla memoria di programma alla CPU.

Logica esterna al microcomputer,
p.e. tastiere, display, stampanti
e unità di memoria di massa
Ingresso dati
Uscita dati
Ingresso dati
Uscita dati
Risultati
Memoria di programma
Istruzioni
CPU
Memoria dati
Dati da elaborare

L'UNITA' CENTRALE DI ELABORAZIONE (CPU)

CPU

Al centro di tutta la logica dei microcomputer c'è l'unità centrale di elaborazione; essa, la CPU, è la logica elettronica che concretizza tutte le operazioni sui dati.

Ciò significa che in varie altre parti del sistema microcomputer voi potete muovere i dati da una locazione ad un'altra, ma potete modificare tali dati solo entro la CPU.

LOGICA SERIALE

Allo scopo di generare la versatilità e la potenza comunemente associate con i computer, la logica CPU deve essere capace di effettuare un gran numero di differenti operazioni; e ciò è effettivamente quello che la CPU può fare. In ogni caso, la CPU può effettuare solamente un'operazione per volta.

AUGENDO
ADDENDO

Consideriamo l'addizione di due numeri; quando due numeri si sommano, essi prendono il nome di "augendo" e addendo. La loro somma avviene tramite la seguente sequenza seriale di eventi:

132 + 47 = 179

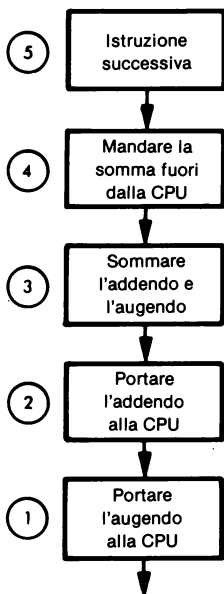
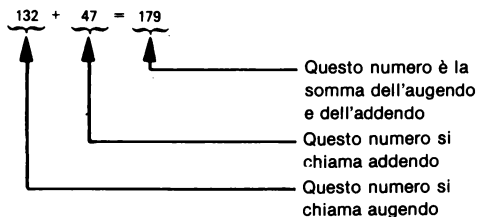
- 179: Questo numero è la somma dell'augendo e dell'addendo
- 47: Questo numero si chiama addendo
- 132: Questo numero si chiama augendo

(5) Istruzione successiva

(4) Mandare la somma fuori dalla CPU

(3) Sommare l'addendo e l'augendo

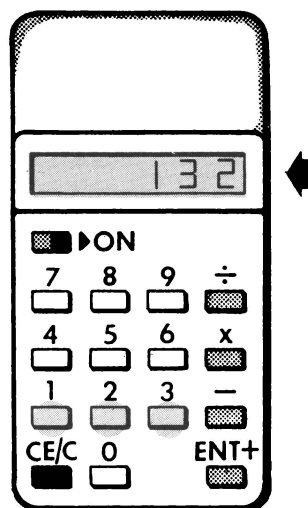
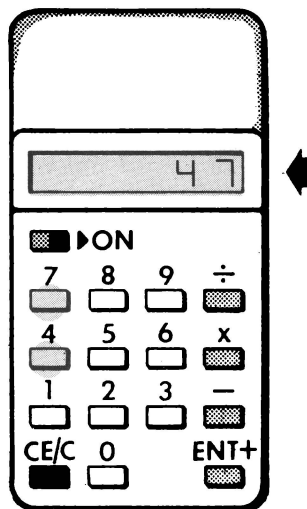
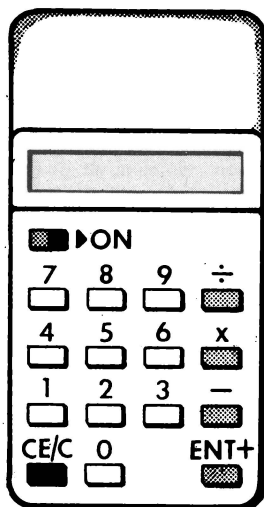
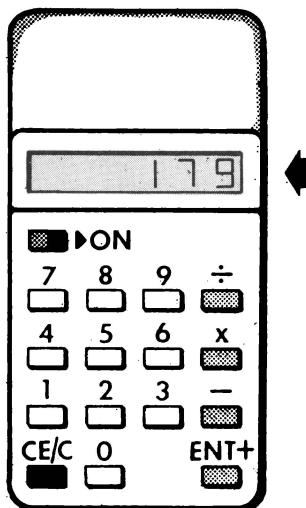
(2) Portare l'addendo alla CPU

(1) Portare l'augendo alla CPU

Ciascun evento è identificato da un numero: (1), (2), (3), (4), ecc.; la CPU effettua ciascun evento come una singola operazione.

Perciò, per realizzare l'addizione qui sopra indicata, la **CPU effettua prima l'evento (1), poi l'evento (2), poi l'evento (3), poi l'evento (4), e cioè:**

- durante la prima fase l'augendo è portato alla CPU;
- durante la seconda fase, l'addendo è portato alla CPU;
- durante la terza fase, addendo ed augendo sono sommati dalla logica elettronica che è entro la CPU;
- durante la quarta fase, la somma viene trasmessa dalla CPU all'uscita.

Queste quattro fasi sono sostanzialmente identiche ai quattro passaggi mediante i quali si possono sommare due numeri usando i più vecchi tipi di calcolatori tascabili.

Nella fase 1, voi battete l'augendo:

Nella fase 2, battete l'addendo:

Nella fase 3, voi premete il tasto +:

La fase 4 si verifica automaticamente; la somma viene portata in uscita, dalla logica del calcolatore, ad un display sul quale voi potete così leggerla:

Ora sapete perché tali calcolatori vi costringevano a fare le cose in modo piuttosto irrazionale; essi vi costringevano ad usare le sequenze logiche del computer.

I più recenti e normali calcolatori usano una logica più complessa che vi permette di operare in sequenze di tipo umano:

- nella fase 1 voi battete l'augendo
- nella fase 2 voi premete il tasto +
- nella fase 3 voi battete l'addendo
- la fase 4 avviene automatica; in uscita c'è la somma.

DISPOSITIVI SERIALI

Possiamo usare i quattro passaggi del calcolatore tascabile (in ciascuna delle due versioni) mediante i quali si sommano due numeri, per illustrare il concetto di un dispositivo seriale, in quanto un calcolatore tascabile ed una CPU sono ambedue dispositivi seriali; ciascuno di essi può eseguire solo un'operazione per volta.
Questo è abbastanza semplice da capire nel caso del calcolatore tascabile; per esempio voi non potete battere simultaneamente i due numeri che vanno sommati.
I due numeri devono essere battuti in modo seriale, cioè uno di seguito all'altro. Nel caso di una CPU, voi non potete portare alla stessa l'addendo e l'augendo simultaneamente; ciascun numero deve essere ricercato mediante una fase indipendente, e le due fasi devono verificarsi una dopo l'altra.

PASSI LOGICI SERIALI

PASSO DI ISTRUZIONE

Il problema successivo che passiamo a trattare è il determinare in che cosa consiste un singolo "passo", o fase. Nel caso del calcolatore tascabile, non si tratta di una considerazione molto importante.

Quando coi tasti inserite il numero 132 l'inserimento di tutto il numero può costituire un "passo"? O è la battuta di ogni singolo tasto a costituire un "passo" individuale? Francamente, per un calcolatore tascabile, questa è una questione trascurabile.
Ma qual'è invece la situazione se dovete scrivere una sequenza di istruzioni che qualcun altro debba poi seguire:

Potreste scrivere il seguente passo singolo:
1) battere 132 alla tastiera.

Potreste anche spezzare questo passo in 3 singole fasi:

1) battere il tasto 1
2) battere il tasto 3
3) battere il tasto 2.

Consideriamo un esempio ancor più "terra a terra": il mangiare una fetta di dolce. Supponiamo che questa fetta di dolce possa essere mangiata in dieci bocconi; si tratta allora di un processo a dieci passi?
Forse, ma forse no. Il mangiare un singolo boccone di dolce può di per se stesso consistere nei seguenti quattro passi:

1) separare un pezzo di dolce con la forchetta;
2) infilare il pezzo così separato con la punta della forchetta;
3) portare alla bocca il pezzo di dolce.
4) masticare ed inghiottire il dolce.

ISTRUZIONI

Sarebbe facile spezzettare ancora questi quattro passi, creando un numero qualsiasi di passi addizionali più piccoli; e questo, come vale quando si tratta di mangiare il dolce, è vero anche per le singole fasi operative di una C.P.U.

Alcune CPU eseguono operazioni in passi relativamente ampi; altre mettono in sequenza eventi come serie di passi relativamente piccoli.
In ogni caso, per qualsiasi CPU, ciascun passo è in modo chiaro e non ambiguo, definito come un'"istruzione".

Non c'è quindi niente di vago in una istruzione individuale o passo singolo, che può essere eseguito da ogni CPU.

SET DI ISTRUZIONI

Ciascuna CPU risponde ad un numero fisso di istruzioni. Queste istruzioni, prese assieme, sono definite col termine "set di istruzioni". Tipicamente, una CPU avrà da 40 a 200 diverse istruzioni in un set.

Ciascuna istruzione è rappresentata da un numero unico, il quale, se trasmesso alla CPU al momento giusto, porta la CPU ad eseguire le operazioni associate con quella istruzione. Per esempio, la nostra sequenza di addizione si può illustrare come segue:

④ Mandare la somma fuori dalla CPU — Questa diventa una singola istruzione rappresentata dal numero 096010049

③ Sommare l'addendo e l'augendo — Questa diventa una singola istruzione rappresentata dal numero 128

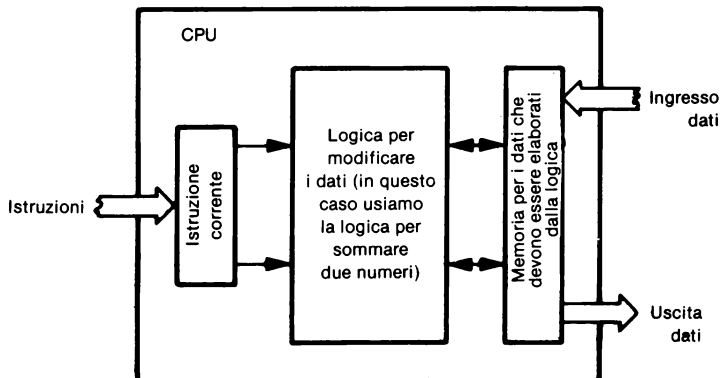
② Portare l'addendo alla CPU — Questa diventa una singola istruzione rappresentata dal numero 156010049

① Portare l'augendo alla CPU — Questa diventa una singola istruzione rappresentata dal numero 156010048

To CPU

MEMORIA DATI LOCALE PER CPU

Le quattro istruzioni mostrate qui sopra illustrano un problema logistico associato con la CPU. La CPU ha spazio di memoria per contenere i dati su cui è in procinto di operare, e ciò è tutto (vedi illustrazione):

CPU
Istruzioni
Istruzione corrente
Logica per modificare i dati (in questo caso usiamo la logica per sommare due numeri)
Memoria per i dati che devono essere elaborati dalla logica
Ingresso dati
Uscita dati

Non potete aspettarvi di lasciare l'augendo, l'addendo e la somma nello spazio di immagazzinamento dati della CPU, in quanto questo spazio quasi certamente vi servirà per l'operazione immediatamente successiva che la CPU dovrà eseguire.

L'augendo, l'addendo e la somma devono perciò avere locazioni permanenti di magazzinaggio da qualche parte al di fuori della CPU, per esempio, nella memoria esterna legge/scrive.

Questo è il perché sono presenti i passi (1) (2) e (4).

MEMORIA DI PROGRAMMA

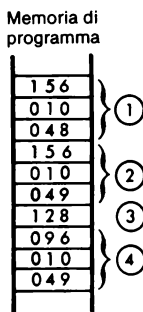
PROGRAMMA

Allo scopo di eseguire qualsiasi operazione, come la somma illustrata, dovete creare una sequenza di istruzioni, che prese assieme costituiscono un programma.

Il programma è una sequenza di numeri; tale sequenza è immagazzinata in una memoria ad accesso rapido, che chiamiamo memoria di programma.

Usando codici numerici arbitrariamente assegnati per le istruzioni di somma, il **programma di somma può essere rappresentato concettualmente come segue:**

Memoria di programma

156
010
048 (1)
156
010
049 (2)
128 (3)
096
010
049 (4)

Il metodo qui usato per illustrare i contenuti della memoria lo vedrete frequentemente in questo, come in altri corsi.

La memoria può essere assimilata ad un alveare di buchi, ognuno dei quali rappresenta una locazione individualmente identificabile ed indirizzabile.

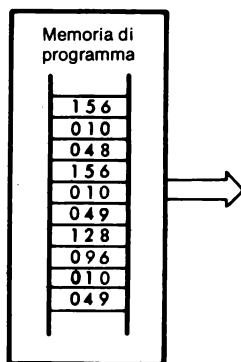
Ogniqualvolta un numero viene trasferito dalla CPU alla memoria, un alveolo verrà riempito. Quando un numero viene trasferito dalla memoria alla CPU, la CPU stessa riceve quanto contenuto nel buco.

LOCAZIONI ED INDIRIZZI DI MEMORIA

Ciascun alveolo è chiamato "locazione di memoria". Ciascuna locazione di memoria è identificabile individualmente mediante un unico indirizzo di memoria.

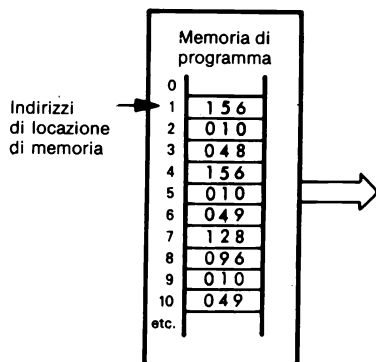
Non ci occuperemo di come creare l'indirizzo di memoria che identifica ogni locazione individuale indirizzabile entro la memoria; perciò la sequenza di istruzioni del programma di addizione qui sopra illustrata sarà rappresentata occupando una sequenza indefinita di locazioni di memoria di programma come segue:

Memoria di programma

156
010
048
156
010
049
128
096
010
049

Senza discutere affatto l'indirizzamento di memoria, potremo illustrare la sequenza d'istruzione del programma di addizione che si verifica nelle prime dieci locazioni indirizzabili della memoria di programma come segue:

Memoria di programma

Indirizzi di locazione di memoria

0	
1	156
2	010
3	048
4	156
5	010
6	049
7	128
8	096
9	010
10	049
etc.	

Non è necessario capire la logica del computer per vedere come le prime dieci locazioni indirizzabili del programma di memoria possono essere riempite con numeri come sopra illustrato.

Occorre invece avere qualche nozione della logica del microcomputer per spiegare come identifichiamo ciascuna di queste, ad ogni altra locazione di memoria. Questo argomento verrà discusso nel 6° capitolo.

MEMORIA DATI

L'informazione che è usata da un programma mentre viene eseguito è definita come un dato. **Nel nostro semplice esempio di addizione, noi andiamo a manipolare tre frammenti di dati: l'augendo, l'addendo e la somma. Questi tre elementi di dati saranno probabilmente immagazzinati in una memoria locale, ad accesso rapido.**

SEQUENZA DEGLI EVENTI DI PROGRAMMA DI ADDIZIONE

Il procedimento di sommare due numeri può ora essere illustrato concettualmente come segue:

Passo 1: estrarre l'augendo

Logica esterna

Logica esterna

Memoria di programma

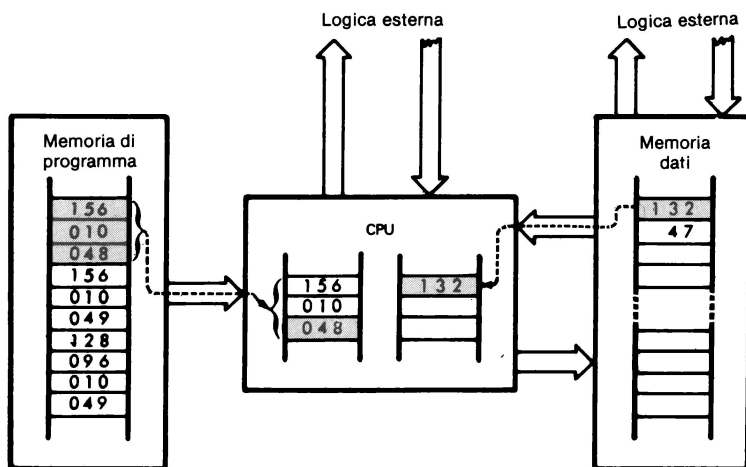
156
010
048
156
010
049
128
096
010
049

CPU

156
010
048

132

Memoria dati

132
47

Passo 2: estrarre l'addendo

Logica esterna

Logica esterna

Memoria di programma

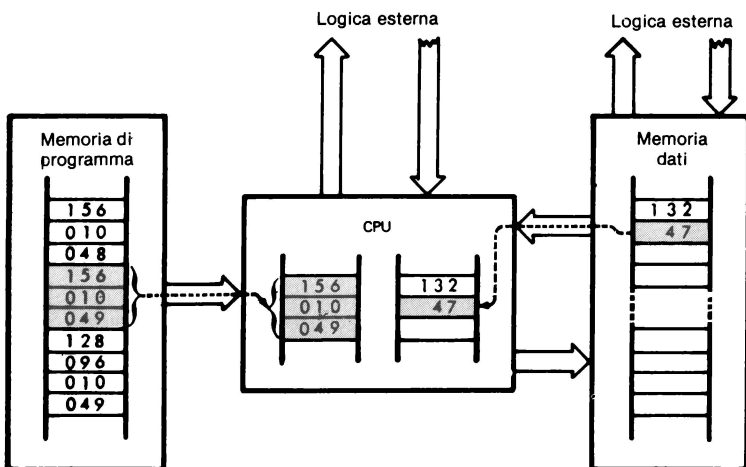
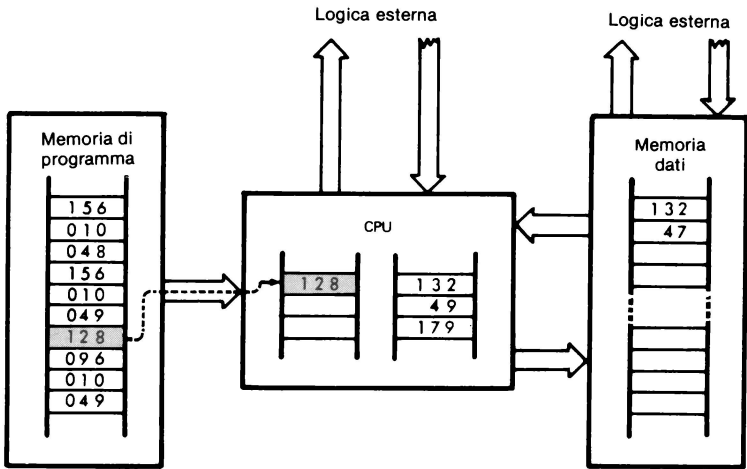
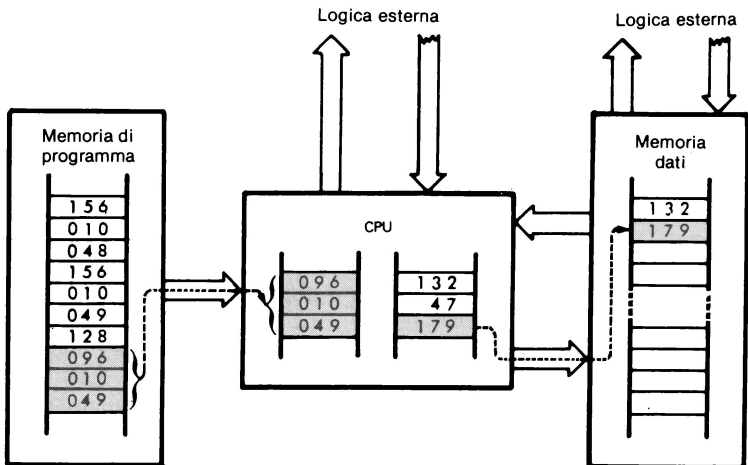
156
010
048
156
010
049
128
096
010
049

CPU

156
010
049

132
47

Memoria dati

132
47

Passo 3: generare la somma

Passo 4: far uscire la somma

Per ciascuno dei quattro passi sopra illustrato, il primo evento a verificarsi sarà il trasferimento di un codice di istruzione della memoria di programma alla CPU.

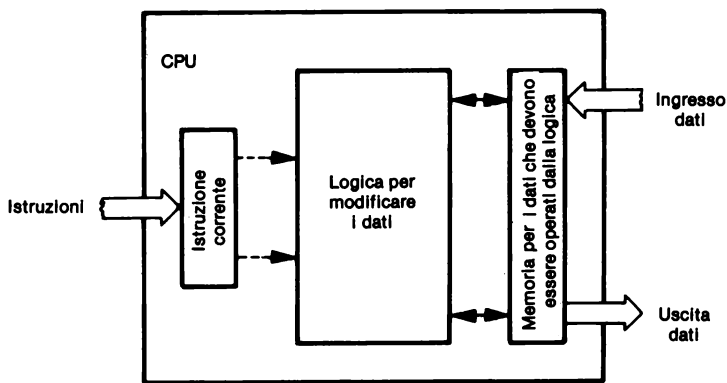
In ciascun passo il codice d'istruzione è il numero cerchiato nella memoria di programma. La CPU non può sapere cosa fare finché il codice d'istruzione non l'ha raggiunta.

Una volta che il codice d'istruzione ha raggiunto la CPU, si verificano concretamente le operazioni richieste dal singolo passo.

Da notare che nel passo 4 la somma è arbitrariamente mostrata riscritta nella stessa locazione di memoria dati dalla quale era stato estratto l'addendo; così che l'addendo andrà perduto.

Capitolo 6
METTIAMO ASSIEME IL TUTTO

Esamineremo ora la logica della CPU.

Nel capitolo 5 avevamo superficialmente illustrato una CPU come segue:

DIMENSIONE DI PAROLA

Nel capitolo 4 abbiamo visto come codici d'istruzione e dati di vari tipi possano diventare tutti serie di bit dall'aspetto identico.

Probabilmente il risultato concettuale più importante di detto capitolo è la conclusione che dobbiamo trarre, e cioè che gli switch a due stati, o bit, non sono di per sè molto utili; invece gruppi di bit possono essere interpretati in modo vario e produttivo.

Perciò la prima e più importante decisione che un progettista di microprocessore deve prendere è di scegliere il numero di bit che il microprocessore stesso manipolerà per ogni volta.

Questo numero lo chiamiamo "dimensione di parola" (word size) del microprocessore.

Si potrebbe progettare un microprocessore che usi tutti i bit che gli servono ad ogni momento; ma nessun microprocessore esistente è realizzato in questo modo.

Tutti i microprocessori esistenti hanno una certa "dimensione di parola" che si applica a ciascun tipo di informazione manipolata.

Facciamo un esempio aritmetico, supponendo che il microprocessore debba addizionare 4213_{10} e 246_{10}.

Ciò si può illustrare come segue:

Decimale		Binario	
		12 11 10 9 8 7 6 5 4 3 2 1 0	← N. di bit
4 2 1 3		1 0 0 0 0 0 1 1 1 0 1 0 1	Augendo
+ 2 4 6	+	1 1 1 1 0 1 1 0	Addendo
= 4 4 5 9	=	1 0 0 0 1 0 1 1 1 0 1 1 1	Somma

I due numeri che dobbiamo sommare richiedono tredici bit per l'augendo, otto bit per l'addendo e tredici bit per la somma.

Un microprocessore potrebbe concepibilmente essere progettato in modo tale da assegnare per l'esempio di somma ora illustrato, tredici switch per augendo e somma ed otto per l'addendo; ma cosa avviene se la somma immediatamente successiva richiede sette switch per l'augendo, quindici per l'addendo e sedici per la somma?

O il microprocessore deve avere una qualche serie sovradimensionata di switch e quindi sprecarne un sacco il più delle volte:

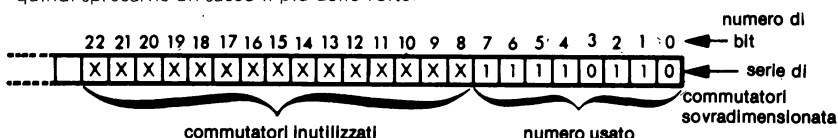
numero di bit

22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

X X X X X X X X X X X X X X X 1 1 1 1 0 1 1 0

serie di commutatori sovradimensionata

commutatori inutilizzati

numero usato

oppure il microprocessore deve riutilizzare gli stessi switch in una grande varietà di modi diversi; e tutto ciò potrebbe diventare incredibilmente complicato:

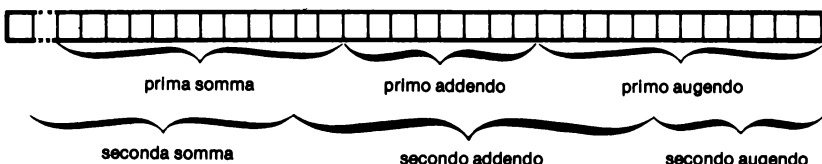
prima somma

primo addendo

primo augendo

seconda somma

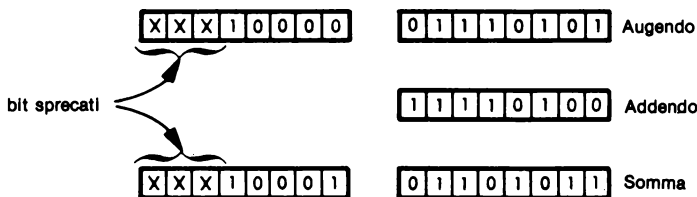
secondo addendo

secondo augendo

Il tipo di logica complessa illustrato sopra non permette di guadagnare alcun vantaggio pratico rispetto all'adozione di una qualche ampiezza fissa di bit per manipolare tutta l'informazione.

Consideriamo otto bit, cioè un byte. Il nostro esempio di addizione sarebbe effettuato, in una unità byte, come segue:

X X X 1 0 0 0 0   0 1 1 1 0 1 0 1 Augendo

bit sprecati   1 1 1 1 0 1 0 0 Addendo

X X X 1 0 0 0 1   0 1 1 0 1 0 1 1 Somma

Ci sono si alcuni bit sprecati, ma la semplificazione della logica è una compensazione più che sufficiente.

La maggior parte dei microprocessori oggi sul mercato elaborano l'informazione in unità di otto bit.

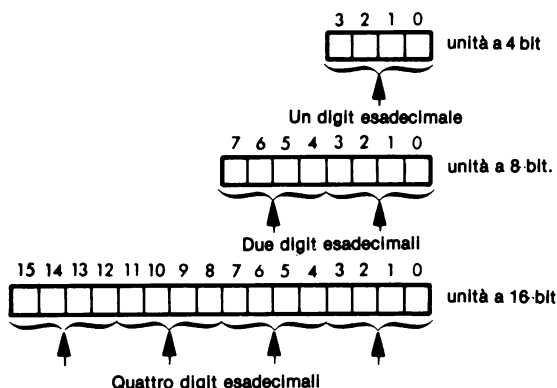
Di conseguenza ci si riferisce ad essi come microprocessori ad 8 bit. Esistono alcuni microprocessori, ormai obsoleti, a 4 bit, mentre alcuni dei più moderni e potenti

elaborano l'informazione in unità a 16 bit.

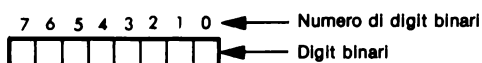
Da notare che queste lunghezze di bit (4, 8 e 16) sono tutte realizzate per permettere un facile conteggio entro un sistema binario:

3 2 1 0 — unità a 4 bit — Un digit esadecimale

7 6 5 4 3 2 1 0 — unità a 8 bit. — Due digit esadecimali

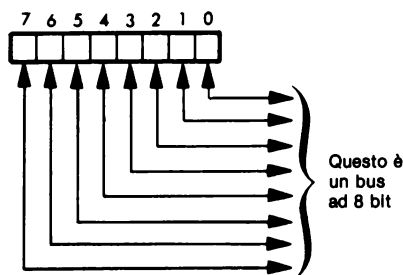
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 — unità a 16 bit — Quattro digit esadecimali

Esaminiamo le implicazioni di una lunghezza fissa di parola. Per un microcomputer ad 8 bit, tutta l'informazione — dati, codici di caratteri e d'istruzione — deve essere immagazzinata in unità di otto cifre binarie. Rappresentiamo questa unità ad otto bit come segue:

7 6 5 4 3 2 1 0 ← Numero di digit binari

Digit binari

I BUS

L'informazione deve anche essere trasmessa da una parte di un sistema microcomputer all'altra. Poiché tutta l'informazione viene elaborata come unità ad 8 bit, i suoi trasferimenti devono verificarsi 8 bit alla volta; perciò questi trasferimenti hanno luogo attraverso otto conduttori in parallelo:

7 6 5 4 3 2 1 0

Questo è un bus ad 8 bit

Questi conduttori li chiamiamo BUS.

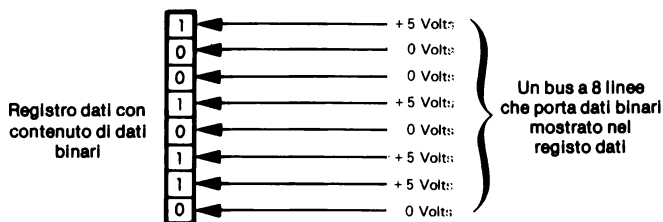
Un bus non è altro che un gruppo di linee elettricamente conduttrici su cui vengono trasferiti dati binari.

Un bit 1, o impulso "on", consiste nella presenza di una tensione sul conduttore;

un bit Ø, o impulso "off" corrisponde all'assenza di qualsiasi tensione sul conduttore. Ciò si può illustrare come segue:

Registro dati con contenuto di dati binari

1	+ 5 Volts
0	0 Volts
0	0 Volts
1	+ 5 Volts
0	0 Volts
1	+ 5 Volts
1	+ 5 Volts
0	0 Volts

Un bus a 8 linee che porta dati binari mostrato nel registo dati

LA RAPPRESENTAZIONE DEI SEGNALI SULLE LINEE DEL BUS

Allo scopo di identificare i livelli di segnale sulle linee del bus ricorriamo ad un artificio grafico. Per una singola linea di bus, disegnamo una linea continua; la linea è alta quando è presente tensione ed è bassa quando non c'è tensione. Ciò può essere rappresentato come segue:

segnale = 1

Segnale = 0

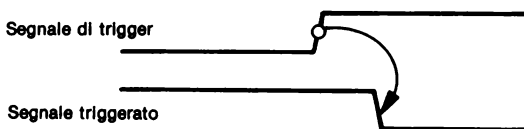
Tempo

Quando un bus ha più di una linea, alcune linee possono essere soggette a tensione mentre altre no; identifichiamo l'istante nel quale una o più linee di bus cambiano stato come segue:

una o più linee di bus possono cambiare stato in questi punti

Talvolta il cambiamento di stato di un segnale fa scattare il cambiamento di stato di un altro segnale; questa azione di trigger si rappresenta come segue:

Segnale di trigger

Segnale triggerato

Non ha importanza a quale segnale cambi il livello; da basso ad alto __/‾‾ può essere sostituito con da alto a basso ‾‾__ e viceversa.

Quello che importa è che un cambiamento di livello di segnale, così indicato:

sia identificato come causa di cambiamento di stato di un altro segnale; come segue:

REGISTRI

Gli switch che contengono l'informazione vengono chiamati "registri".

L'informazione viene trasmessa da e per i registri tramite i bus. Ora possiamo vedere immediatamente che un microprocessore a 4 bit sarà più semplice di uno a 8 bit, in quanto ciascun registro o bus richiede solo quattro switch o linee, anziché otto.

Analogamente, un microprocessore a 16 bit ha una logica più complicata di uno a 8, in quanto ciascun registro o bus richiede 16 switch o linee, anziché otto.

L'UNITA' ARITMETICO LOGICA

ALU

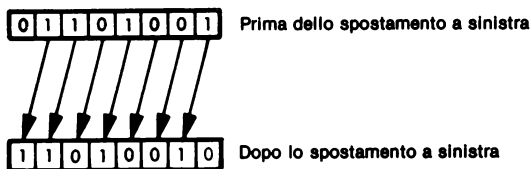
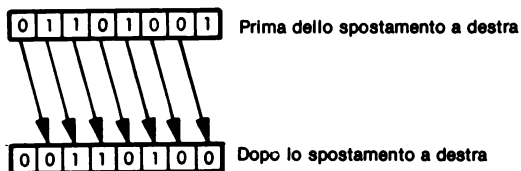
La dimensione di parola di un microprocessore si applica anche alla logica che gestisce la manipolazione dei dati. Le diverse operazioni aritmetiche e logiche descritte nel capitolo 4 verranno tutte eseguite su unità di dati con dimensioni fisse, uguali alla dimensione di parola del microprocessore.

Così per un micro a 8 bit, si avranno sempre dati in input ad 8 bit per ogni operazione sia aritmetica che booleana, e si avrà quindi un risultato ad 8 bit.

Tutta questa logica è collegata assieme in una locazione cui ci si riferisce col nome di Unità Aritmetica e Logica.

Tale unità, ALU, deriva il suo nome dal fatto che essa segue sia operazioni aritmetiche (cioè somma e sottrazione) sia operazioni logiche, le principali delle quali sono quelle di tipo Booleano, AND, OR, XOR e NOT.
La fig. 6-1 fornisce l'illustrazione funzionale della logica di una ALU. **In aggiunta all'esecuzione delle operazioni aritmetiche e Booleane descritte nel capitolo 4, una ALU può anche essere in grado di operare lo shift di dati a sinistra:**

0	1	1	0	1	0	0	1	**Prima dello spostamento a sinistra**
1	1	0	1	0	0	1	0	**Dopo lo spostamento a sinistra**

e/o shift a destra:

0	1	1	0	1	0	0	1	**Prima dello spostamento a destra**
0	0	1	1	0	1	0	0	**Dopo lo spostamento a destra**

I bit 0 e 1 mostrati nelle figure precedenti sono stati scelti arbitrariamente e non hanno alcun particolare significato.

A noi non interessa il metodo vero e proprio usato per creare la logica di addizione — o qualunque altro tipo di logica — entro l'ALU. Questo tipo di dettaglio serve solo a quei pochi che progettano microprocessori.

Ciò che è degno di nota è il fatto che la ALU possiede uno o più bus che le portano i dati in entrata, ed uno o più bus attraverso i quali i dati vengono trasmessi all'esterno. (La ALU potrebbe anche avere uno o più bus interni, come si vede dalla figura 6-1,

ma questi non ci interessano).

I bus saranno quindi: bus a 4 bit per i microprocessori a 4 bit, bus a 8 bit per i microprocessori ad 8 bit, bus a 16 bit per i microprocessori a 16 bit;

ALU

Bus a 8 linee

Commutatori di status flag

Logica di spostamento

Logica NOT

Logica AND, OR, XOR

Logica di addizione

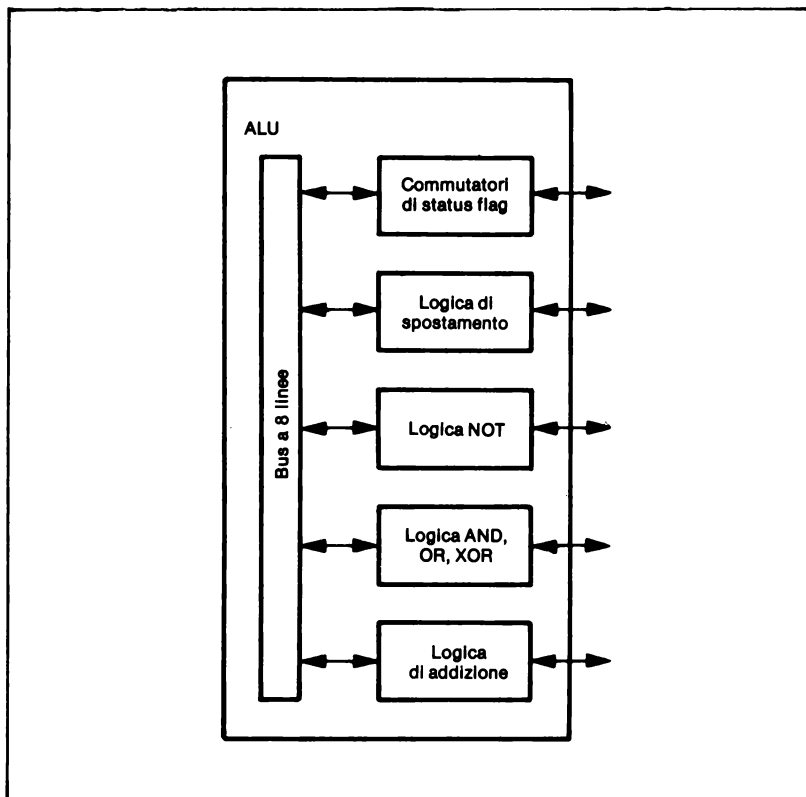



Figura 6-1. Unità aritmetica logica ALU.

L'unico aspetto veramente importante di fig. 6-1 è il fatto che una ALU deve essere presente, e che essa esegue effettivamente le manipolazioni dei dati che possono essere richiesti.

Affinché queste manipolazioni di dati si verifichino concretamente, i dati devono essere trasferiti dal registro dati, tramite bus appropriati, alla ALU. I risultati di ciascuna manipolazione dati devono essere trasferiti, mediante un bus appropriato, indietro al registro dati.

Cosicché l'unica "magia" che resta inspiegabile in ciascuna operazione di computer è in concreto la procedura mediante la quale si effettuano le operazioni entro la ALU.

Noi non andremo a descrivere la logica interna della ALU, poiché ci sono probabilmente tanti modi di farlo quanti sono i progettisti di logica; inoltre, per voi, come utenti di microprocessori, l'argomento non è di grande interesse. Basta ricordare che, non importa quanto sia complicata l'operazione che volete far eseguire al computer, essa alla fin fine risulterà spezzettata in una sequenza seriale di passi, ciascun

dei quali provoca l'invio di dati di ingresso binari alla ALU e la ricezione di risultati in dati binari dalla stessa.

Poiché la ALU riceve gli operandi dai registri dati e ritorna i risultati ad altri registri dati, dobbiamo aggiungere opportuni registri dati alla ALU.

Logica della CPU dal lato dati della ALU

ALU

Bus a 8 linee

Status flag

Logica di spostamento

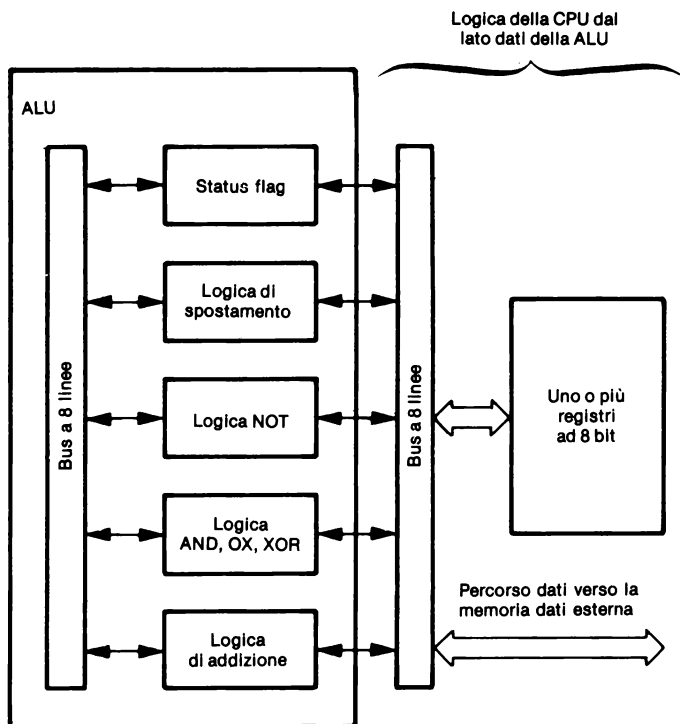
Logica NOT

Logica AND, OX, XOR

Logica di addizione

Bus a 8 linee

Uno o più registri ad 8 bit

Percorso dati verso la memoria dati esterna

UNITA' DI CONTROLLO

Dall'altro lato della ALU, ci serve un registro che contenga le istruzioni che determinano quale parte della ALU sarà usata, e come sarà usata. La determinazione di "quale" e "come" è fatta da un nuovo blocco di logica che chiamiamo "unità di controllo".

Percorso dati del bus a 8 bit dalla memoria di programma

Registro istruzioni

Unità di controllo

Logica CPU dal lato programmi della ALU

Segnali di controllo

Logica CPU dal lato dati della ALU

ALU

Bus a 8 linee

Status flag

Logica di spostamento

Logica NOT

Logica AND, OX, XOR

Logica di addizione

Bus 8 linee

Uno o più registri ad 8 bit

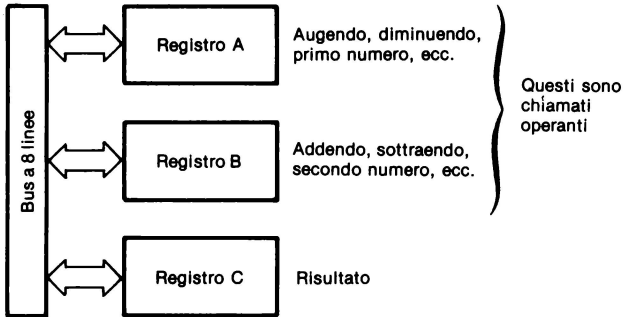
Percorso dati del bus a 8 bit verso la memoria dati

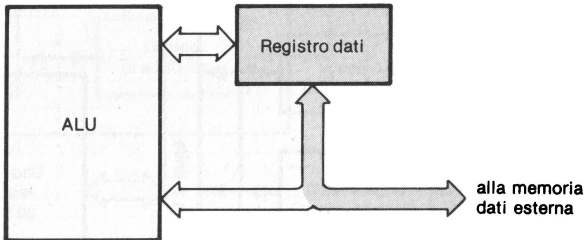
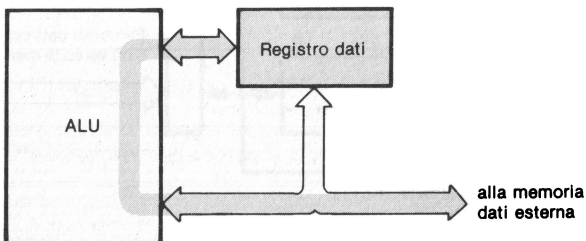
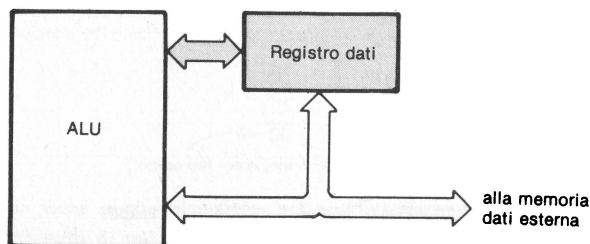
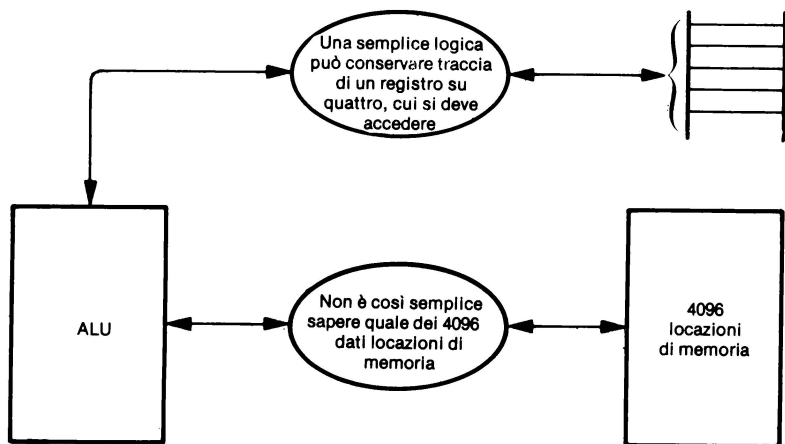
LOCICA ADDIZIONALE DELLA CPU

Esaminiamo ora la logica da ciascun lato della ALU.

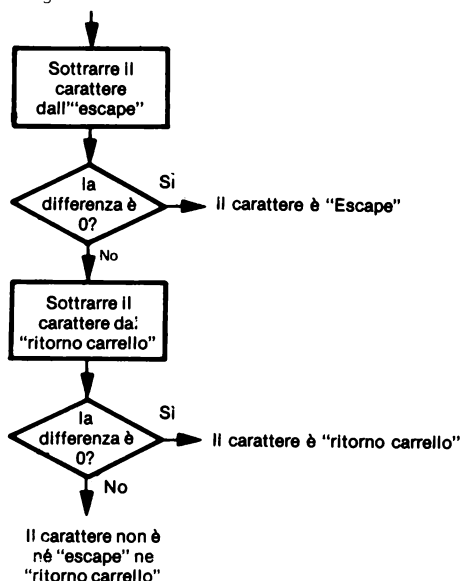
REGISTRI DATI

OPERANDI

Dalla parte dati della ALU ci serve uno o più registri per contenere i dati che stanno per essere operati dalla ALU. Non esiste alcun numero di registri dati "dettato dal buon senso" che ogni singolo progettista possa includere dalla parte dati della ALU; si potrebbe per esempio decidere che tre registri sono una buona idea, in quanto molte operazioni aritmetiche e Booleane usano due operandi per fornire un risultato:

Ma spezzando l'operazione aritmetica e Booleana in un certo numero di passi **si potrebbe risolvere con un solo registro dati.** Questi sono i passi necessari:

1) Portare il primo operando al registro dati:

2) Effettuare ciascuna operazione di ALU, prelevando un operando dal registro dati e l'altro operando direttamente dalla memoria esterna:

3) Riporre il risultato nel registro dati:

Ma esistono anche buoni argomenti per avere più di tre registri dati dalla parte dati della ALU.
I dati possono essere trasferiti fra un registro e la ALU molto più velocemente di quanto lo passano fra la memoria dati esterna e la ALU.
Ciò avviene in quanto nella CPU vi sono molto pochi registri dati, talchè si può immediatamente identificare il registro cui si vuole accedere.
C'è invece un gran numero di locazioni di memorie dati esterne, il ché significa che ogni qualvolta la CPU accede ad una locazione di memoria dati esterna, ci sarà coinvolta una fase completa di identificazione di locazione di memoria.
La CPU deve cioè spendere tempo per individuare esattamente a quale locazione di memoria dati si suppone, si debba accedere. Questo si può presentare come segue:

L'UTILIZZO DEI REGISTRI DATI

Avendo molti registri dati, un microprocessore permette di mantenere all'interno della CPU ognuno dei dati cui si deve più frequentemente accedere. Per decidere sul numero ottimale di registri dati per una CPU, ci occorre in qualche modo avere compreso la funzione per la quale i registri dati servono .
Lasciamo quindi da parte, per un po' la trattazione attuale, per esaminare un esempio concreto su come vengono usati i registri dei dati.
Nel programma contabile di Joe Bitburger, ogni volta che Joe inserisce un carattere mediante la tastiera, il programma verifica se il carattere è un "escape" o un "ritorno carrello".
Ora alcuni microcomputer non hanno un'istruzione apposita per questa verifica ("test"); in mancanza di essa, **si può realizzare un testo sottraendo il codice del**

carattere in arrivo dal carattere "escape" e poi dal carattere "ritorno carrello", secondo la logica di figura:

Rappresenteremo un "ritorno carrello" quale carattere di tastiera usando il simbolo (Cr)

Ma ha senso sottrarre una lettera dall'alfabeto da un ritorno carrello?

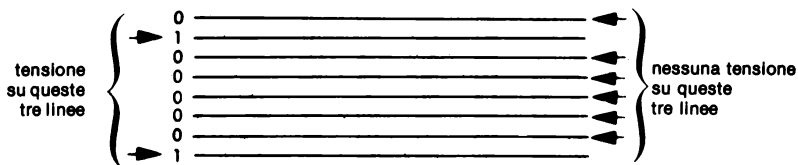
(Cr) − A = ?

INTERPRETAZIONI MULTIPLE DEI DATI BINARI

Per la logica umana questa sottrazione non ha senso; ma occorre ricordare che i microcomputer rappresentano i caratteri mediante sequenze di digit binari, ed anche i numeri li rappresentano mediante le stesse sequenze. Ricorderete quando frequentemente, nel capitolo 4, sia stato detto che occorre ricordare come si debba interpretare una sequenza di digit binari. Questo aveva i suoi vantaggi, perché ora c'è qualcosa da ricordare.

Supponiamo che Joe prema il tasto A della sua tastiera; il microcomputer vede arrivare dalla tastiera la seguente sequenza binaria:

01000001

Questa serie di bit arriverà in un bus a 8 linee come segue:

Per quanto concerne il microcomputer, questa non è altro che una sequenza di digit binari. Se essa viene conservata in memoria, e riutilizzata più tardi, potrà essere inviata ad un display video per riprodurre la lettera A; il video display infatti è progettato per interpretare tutte le sequenze di bit in arrivo come caratteri ASCII.

Ma all'interno della CPU una serie di bit è semplicemente una serie di bit; è per questo che la sequenza di bit della lettera A può essere sottratta da quella del ritorno carrello, trattando ciascuna di esse come puri e semplici dati binari:

Cr 00001101
- A - 01000001
CC 11001100

Codice del carattere ASCII

Intanto che non viene scritto nient'altro nei registri che contengono i caratteri A e (Cr) , il contenuto di tali registri viene conservato inalterato. Quando, successivamente, si accede a questi registri, se ne può interpretare il contenuto come caratteri ASCII. Per esempio, si può trasmettere il codice del carattere A al video display per generare una risposta esattamente dopo aver sottratto il codice del carattere A dal ritorno carrello. Possiamo ora rappresentare l'intera sequenza di istruzioni che riceve e verifica i caratteri in arrivo, usando questo grafico di flusso più complesso.

Caricaro 00011011 che rappresenta il carattere "escape" nel registro A

Caricare 0001101, che rappresenta il ritorno carrello nel registro B

Memorizzare il codice di carattere della tastiera nel registro C

Sottrarre il contenuto del registro C dal contenuto del registro A; questo equivale a "escape" A.

la differenza è 0 ?

Si → Il carattere è "escape"

No

Sottrarre il contenuto del registro C dal contenuto del registro A; questo equivale a "ritorno carrello" A.

la differenza 0?

Si → Il carattere è "ritorno carrello"

No

Trasmettere il contenuto del registro C al display video; questo è l'eco di A.

FLAG DI STATO ZERO

Seguendo ogni sottrazione, si userà uno status flag per determinare se il carattere è un codice di ritorno carrello o di escape. Pressoché tutti i microprocessori hanno uno status flag contrassegnato col termine di stato zero; questo flag è utilizzato per rivelare i risultati zero delle operazioni, come segue:

se il risultato è 0, il flag di stato zero è 1;
se il risultato non è 0, il flag di stato è 0.

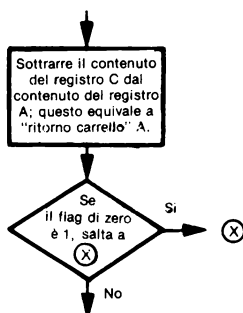
Secondo questa logica distorta, il flag di stato zero è sempre 1 per risultati eguali a zero, ed è sempre 0 per risultati diversi da zero.

Con ciò possiamo costruire la nostra logica di prova caratteri, mediante la sequenza di istruzioni che segue:

Sottrarre il contenuto del registro C dal contenuto del registro A; questo equivale a "ritorno carrello" A.

Se il flag di zero è 1, salta a ⊗

Si ⊗

No

⊗ è l'istruzione da eseguire se dalla tastiera viene inserito un ritorno carrello anziché un A. Questa istruzione viene scelta e identificata mediante il suo indirizzo di memoria di programma, cioè dall'indirizzo della locazione della memoria di programma in cui è contenuto il codice di istruzioni.

Con tutto ciò ci siamo leggermente discostati dalla discussione che avevamo sottomano: il numero corretto di registri necessari sul lato dati della ALU.

Ma ora che abbiamo capito come sono usati i registri dei dati, siamo in grado di giustificare un qualche numero "corretto" di registri dati che qualsiasi microprocessore dovrebbe avere?

Disgraziatamente, non esiste un tale numero esatto, poiché ciascun microprocessore è stato progettato secondo una propria struttura.

Quanta della logica disponibile il progettista ha "messo da parte" per i registri e quanta per le altre cose?

Ricordiamo che occorrono otto switch per ciascun registro, più le connessioni fra registri e bus all'interno del microprocessore; ma i medesimi switch e bus possono essere sfruttati in un'ampia varietà di modi.

MEMORIA DELLA CPU

Normalmente, sul lato dati della ALU, si possono trovare da 4 a 8 registri; **ma il numero può, almeno teoricamente, variare da 1 a 32. Ci sono anche alcuni microprocessori che hanno una piccola parte di memoria dati entro la CPU.**

Queste locazioni di memoria possono essere 64 o più, e sono qualcosa in più delle locazioni esterne di memoria dati e qualcosa in meno dei registri dati veri e propri.

IL REGISTRO ISTRUZIONI E L'UNITA' DI CONTROLLO

Nella zona memoria di programma della ALU, ci deve essere un singolo registro nel quale contenere il codice binario che rappresenta l'istruzione da eseguirsi correntemente; è questo che viene chiamato "registro istruzioni".
Supponiamo che l'istruzione da eseguire sommi il contenuto dei registri dati A e B, per poi ritornare la somma al registro dati A (per il momento supponiamo che i registri A e B esistano nella zona dati della CPU).

L'appropriato codice binario di istruzioni sarà fornito (sotto forma di dati) alla CPU, dove verrà immagazzinato nel registro istruzioni.

I contenuti di tale registro agiscono, nei confronti di un blocco di logica chiamato "unità di controllo", come uno dei 256 trigger; ci sono 256 possibili trigger in quanto esistono 256 combinazioni di 0 e 1 che si possono ottenere da otto bit.

Un microprocessore a 16 bit, avendo un registro istruzioni di 16 bit, presenterà 65.536 combinazioni. Il registro istruzioni "triggera" la partenza di una sequenza di segnali in uscita dalla unità di controllo e ciò abilita i trasferimenti dati e le operazioni della ALU richiesti dall'istruzione da eseguire.

Ben poco c'è da sapere sulla unità di controllo, sul registro istruzioni e sul modo in cui essi interagiscono; l'unità di controllo produce segnali di controllo che muovono i dati verso le zone in cui si suppone debbano essere spostati ed abilita la logica della ALU nei tempi previsti. Per i microcomputer normalmente reperibili sul mercato, non c'è nulla che voi, come utenti, possiate fare attorno all'unità di controllo; per esempio non è possibile modificare il modo in cui essa risponde ad un codice di istruzione.

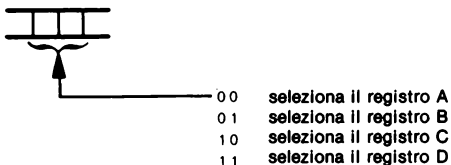
L'unità di controllo agisce come unità di accoppiamento fra un'istruzione e gli eventi che si verificano quando l'istruzione viene eseguita; ciò è tutto quanto serve sapere al proposito.
E' invece importante conoscere la dimensione del registro istruzioni.
Un registro istruzioni ad 8 bit può specificare solo 256 operazioni diverse, o variazioni su operazioni.

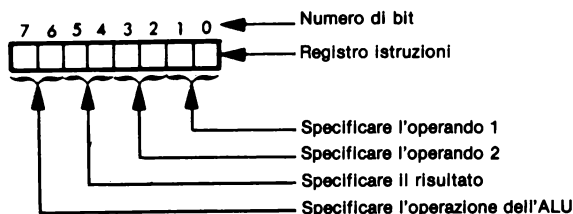
Ciò avviene perché ci sono solo 256 combinazioni di digit 0 e 1 che si possono ottenere da otto digit binari. 256 può sembrare anche un numero elevato, ma in realtà non lo è.

Consideriamo le operazioni eseguite dalla ALU; dentro questo blocco logico abbiamo definito esattamente sette operazioni:

shift a sinistra
shift a destra
complementazione
AND
OR
OR esclusivo
somma.

Anche se si tratta di sole sette operazioni dell'ALU, le istruzioni devono definire molto di più, e cioè le sorgenti degli operandi e la destinazione del risultato.
Supponendo di avere quattro registri di dati (A, B, C e D), servono due degli otto bit di codice istruzioni ogniqualvolta si debba identificare uno dei quattro registri.

00 **seleziona il registro A**
01 **seleziona il registro B**
10 **seleziona il registro C**
11 **seleziona il registro D**

In altre parole, un codice di istruzione deve essere assolutamente specifico. Se si hanno quattro opzioni per i registri, occorrono quattro codici a digit binari separati e distinti per specificare le opzioni prescelte. **Cosicché il codice istruzioni ad 8 bit può essere rappresentato come in figura.**

7 6 5 4 3 2 1 0 — **Numero di bit**
Registro istruzioni
Specificare l'operando 1
Specificare l'operando 2
Specificare il risultato
Specificare l'operazione dell'ALU

Potremo assumere arbitrariamente che, per ciascuna delle specifiche dei tre registri, l'unità di controllo interpreti i bit del codice di istruzioni come segue:

00 = selezione del registro A
01 = selezione del registro B
10 = selezione del registro C
11 = selezione del registro D

I bit del registro istruzioni 6 e 7 possono essere interpretati:

7 6
0 0 = somma
0 1 = AND
1 0 = OR
1 1 = OR esclusivo

Riportiamo un esempio di come si potrebbe interpretare un codice di istruzioni:

7 6 5 4 3 2 1 0 — **Numero di bit**
0 0 1 0 0 1 0 0 — **Registro istruzioni**
Il contenuto del Registro Dati A
e il contenuto del Registro Dati B
saranno aggiunti alla somma
immagazzinata nel Registro Dati C

Abbiamo accuratamente illustrato il modo in cui l'unità di controllo decodifica i contenuti del registro istruzioni; ma l'illustrazione, seppure concettualmente precisa, non è pratica. La serie di bit illustrata si riferisce solamente a quattro delle sette operazioni della ALU, cioè le quattro che richiedono due operazioni di ingresso.

Ma queste quattro operazioni, assieme alle loro opzioni, sfruttano tutti i 256 codici istruzioni; non restano quindi codici disponibili né per alcun'altra operazione della ALU, né per alcuna di quelle istruzioni che non usano la ALU.

E' chiaro quindi che 256 possibili combinazioni di istruzioni non sono gran cosa. Nel volume 1° esploreremo il sistema con cui i progettisti di microprocessori distribuiscono il numero limitato di opzioni di codice istruzioni fra tutti i tipi di istruzioni che possono essere presenti, fornendo una limitata capacità per ciascuna classe di istruzioni; questo procedimento è di facile comprensibilità perché è quasi lo stesso di quando si dispone di un limitato bilancio.

Infatti, se non si ha abbastanza denaro per fare tutto ciò che si vorrebbe, ci si limita un po' in tutti i settori, in modo da trovare il miglior equilibrio fra modo di vita ed entrate.

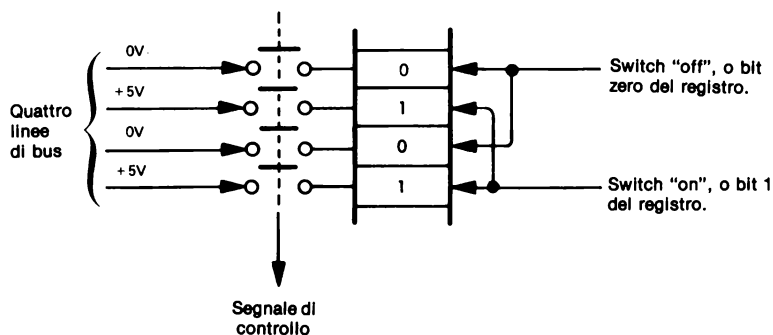
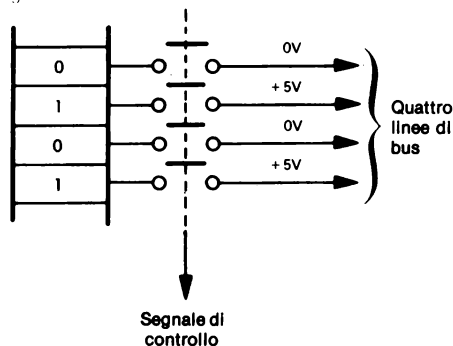
CONCETTI LOGICI E TEMPORIZZAZIONE

Anche se non è importante conoscere le esatte modalità operative di una unità di controllo, vi sono alcuni concetti logici che sono importanti in quanto essi sono universalmente presenti nei sistemi a microcomputer, ora esamineremo quindi tali concetti.

Ogni sequenza di operazioni logiche all'interno della CPU (o di qualsiasi altra parte di un microcomputer) consisterà nel muovere e modificare dati binari.
Entro la CPU, i dati binari devono muoversi fra registri dati, ALU e logica esterna; essi possono invece essere cambiati solo entro la ALU.
In qualsiasi altra parte di un sistema microcomputer, i dati binari verranno spostati fra i registri, e saranno modificati da logiche più semplici della ALU, ma ad essa simili.

LOGICA PER MUOVERE DATI BINARI

SEGNALE DI CONTROLLO

Allo scopo di spostare dati binari, deve essere prodotto un segnale di controllo che agisca da elemento connettore fra lo switch di un registro e la linea di un bus. Questo segnale di controllo è talvolta chiamato "gate signal". Si possono sottoporre a gate dei dati da linee di bus ad un registro:

La posizione di switch (o bit del registro) può essere sottoposta a gate su una linea di bus come segue:

TEMPORIZZAZIONE DI ISTRUZIONI

Durante queste operazioni di trasferimento dati, è molto importante la temporizzazione, in quanto occorre un tempo finito affinché i livelli di tensione appaiano o scompaiano sulle linee del bus; e finché su tali linee esiste un livello fisso, la logica non può collegare le linee del bus ad alcun nuovo switch. Inoltre, la sequenza di eventi corrispondenti all'esecuzione di ogni istruzione deve verificarsi in un qualche ordine ben preciso. Consideriamo il nostro esempio di addizione; la necessaria sequenza di eventi entro la CPU sarà la seguente:

passo 1) portare il codice di istruzioni nel registro istruzioni

8 bit di bus
dati dalla memoria
di programma.
Registro
istruzioni
Unità
di
controllo
Segnali
di controllo
Logica della CPU
sulla parte dati dell'ALU
ALU
Bus a 8 linee
Flag di stato
Logica di
shift
Logica NOT
Logica
AND, OR, XOR
Logica di
addizione
8 line Bus
Registro A
Registro B
Registro C
Registro D
8 bit di bus dati
verso la memoria dati

passo 2) tramite gate, porre il contenuto del registro A sul bus

8 bit di bus
dati dalla memoria
di programma.

Registro
istruzioni

Unità
di
controllo

Segnali di controllo

Logica della CPU sulla
parte dati dell'ALU

ALU

Bus a 8 linee

Flag di stato

Logica di
shift

Logica NOT

Logica
AND, OR, XOR

Logica di
addizione

Bus a 8 linee

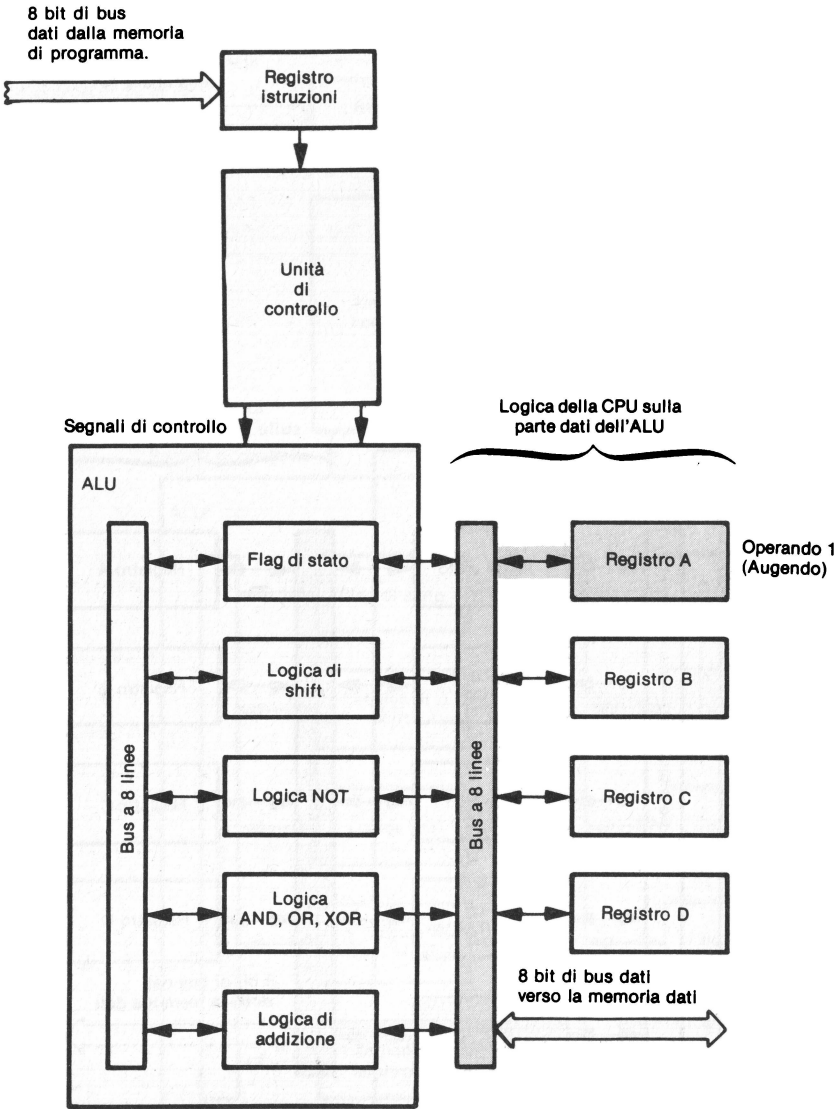
Registro A

Operando 1
(Augendo)

Registro B

Registro C

Registro D

8 bit di bus dati
verso la memoria dati

passo 3) effettuare gate del bus verso la logica addizionale

8 bit di bus dati
dalla memoria
di programma

Registro
istruzioni

Unità
di
controllo

Segnali di controllo

Logica della CPU
sulla parte dati dell'ALU

ALU

Bus a 8 linee

Flag di stato

Logica di
shift

Logica NOT

Logica
AND, OR, XOR

Logica di
addizione

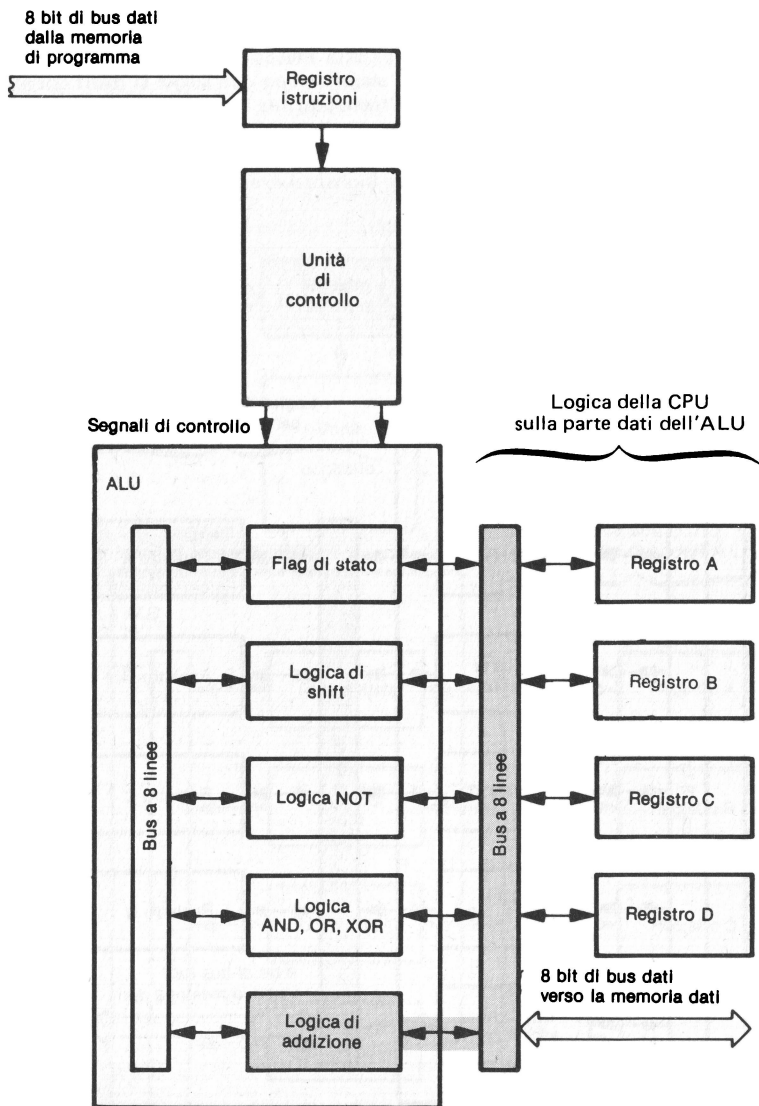
Bus a 8 linee

Registro A

Registro B

Registro C

Registro D

8 bit di bus dati
verso la memoria dati

passo 4) tramite gate, portare il contenuto del registro B sul bus

8 bit di bus dati dalla memoria di programma.

Registro istruzioni

Unità di controllo

Segnali di controllo

Logica della CPU sulla parte dati dell'ALU

ALU

Bus a 8 linee

Flag di stato

Logica di shift

Logica NOT

Logica AND, OR, XOR

Logica di addizione

Bus a 8 linee

Registro A

Registro B

Registro C

Registro D

8 bit di bus dati verso la memoria dati

passo 5) sommare

8 bit di bus dati
dalla memoria
di programma

Registro
istruzioni

Unità
di
controllo

Segnali di controllo

Logica della CPU sulla
parte dati dell'ALU

ALU

Bus a 8 linee

Flag di stato

Logica di
shift

Logica NOT

Logica
AND, OR, XOR

Logica di
addizione

Bus a 8 linee

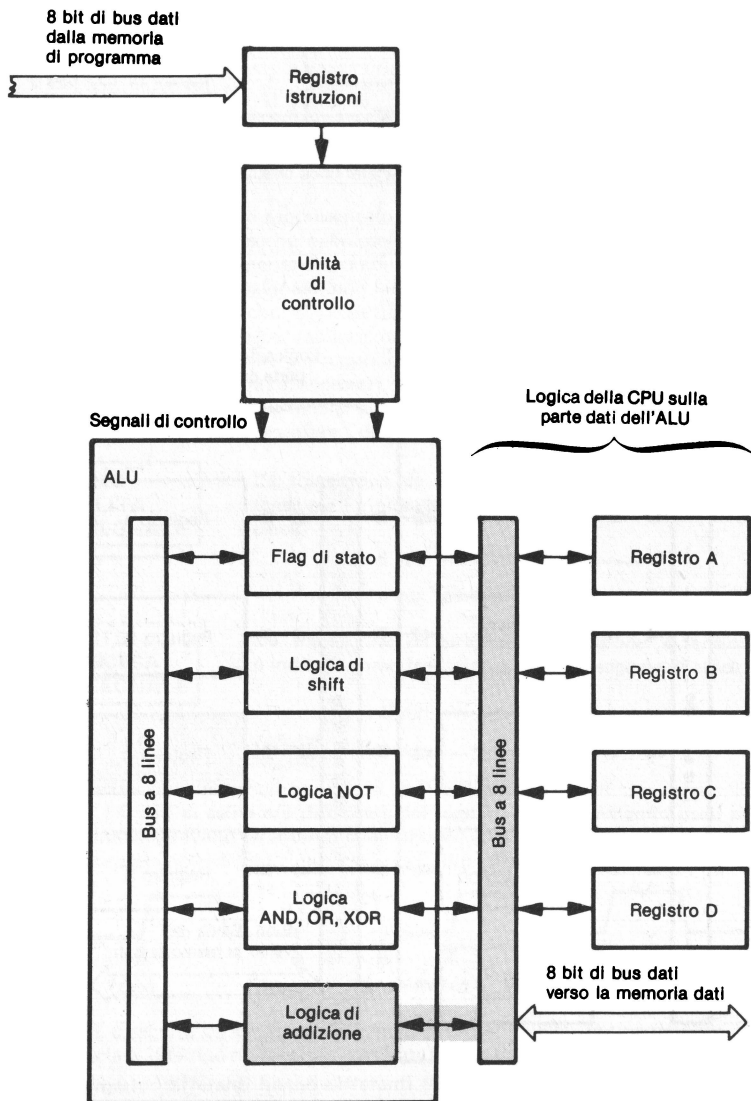
Registro A

Registro B

Registro C

Registro D

8 bit di bus dati
verso la memoria dati

passo 6) effettuare gate della somma sul bus

passo 7) effettuare gate della somma nel registro C

8 bit di bus dati
dalla memoria
di programma

Registro
istruzioni

Unità
di
controllo

Segnali di controllo

Logica della CPU sulla
parte dati dell'ALU

ALU

Bus a 8 linee

Flag di stato

Logica di
shift

Logica NOT

Logica
AND, OR, XOR

Logica di
addizione

Bus a 8 linee

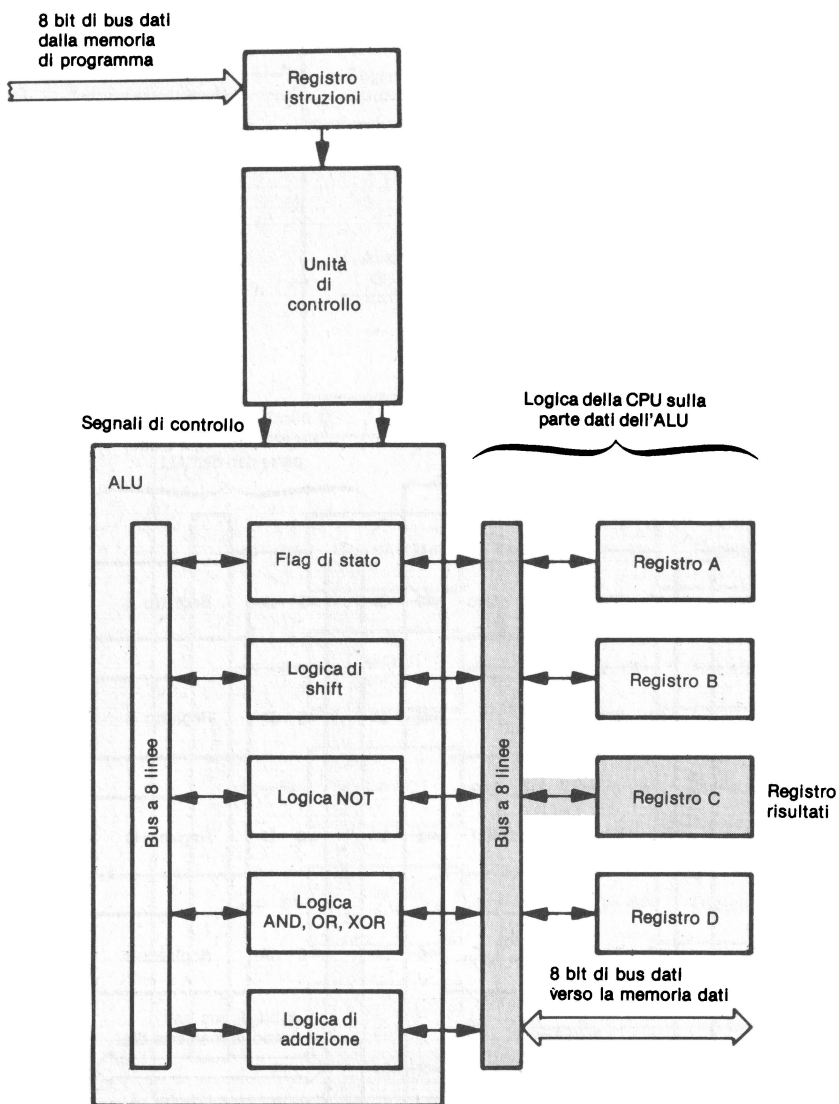
Registro A

Registro B

Registro C

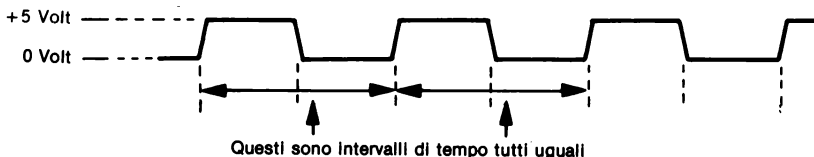
Registro
risultati

Registro D

8 bit di bus dati
verso la memoria dati

IL SEGNALE DI CLOCK ED I TEMPI DI ESECUZIONE DELLE ISTRUZIONI

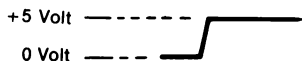
Sequenze di eventi quali i sette passi appena illustrati vengono regolate da un segnale di clock, cosi chiamato in quanto esso genera impulsi di tensione regolari e periodici; un segnale di clock lo possiamo rappresentare come segue:

+5 Volt
0 Volt

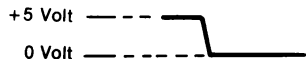
Questi sono intervalli di tempo tutti uguali

Questo segnale di clock è rappresentato in commutazione fra 0 e 5 volt. Non ci sono particolari ragioni intrinseche nella scelta di questi due valori, se non che tali livelli di tensione sono normalmente disponibili nei circuiti dei microcomputer.
Da un punto di vista funzionale, non farebbe alcuna differenza quali tensioni fossero scelte. Le necessità logiche di progetto del chip richiedono che vengano usati un paio di valori di tensione per rappresentare un segnale di clock; gli stessi livelli di tensione si possono anche usare per rappresentare i livelli dei digit 0 e 1.
I tipi più antiquati di microprocessori usano più di due livelli di tensione: i valori addizionali non fanno che rendere più semplice il progetto del chip, ma non prendono parte attiva nel rappresentare i digit binari 0 e 1.

FRONTE DI SALITA DEL SEGNALE

La transizione da tensione zero ad un qualsiasi valore di tensione è indicata come fronte di salita del segnale di clock:

+5 Volt
0 Volt

FRONTE DI DISCESA DEL SEGNALE

La transizione da presenza di tensione a tensione zero è indicata come fronte di discesa del segnale di clock:

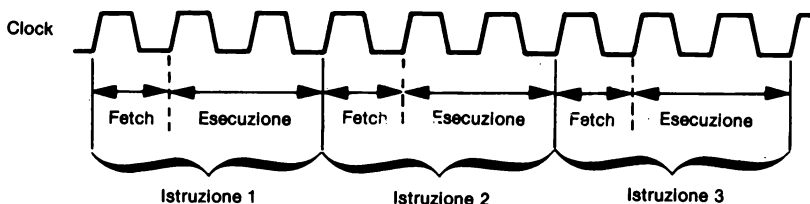
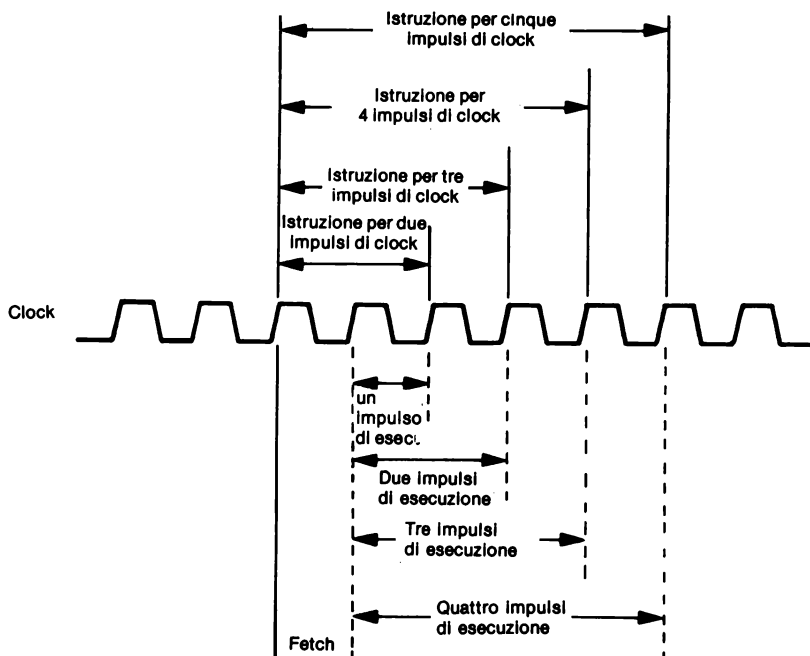
+5 Volt
0 Volt

Naturalmente tutti i segnali hanno un fronte di salita e di discesa, non solo quelli di clock. **I fronti di salita e/o di discesa del segnale di clock vengono usati per scandire il tempo degli eventi all'interno della CPU ed attraverso il sistema microcomputer.** Per il nostro esempio di addizione, ciò si può illustrare come segue:

Passo 1 Passo 2 Passo 3 Passo 4 Passo 5 Passo 6 Passo 7

Il passo 1 è seguito da un tempo maggiore che non gli altri, in quanto alla CPU deve essere lasciato il tempo di decodificare il contenuto del registro istruzioni.

Microcomputer differenti hanno differenti tipi di segnali di clock, dai più semplici ai più complessi. Possono anche esserci più di un segnale di clock, nel qual caso i vari segnali saranno in qualche ben preciso rapporto fra di loro.
Ma, **in qualsiasi caso, è il segnale di clock che agisce come artefice principale delle sequenze di eventi per tutta la logica.**

FETCH DI UNA ISTRUZIONE

Diamo ancora un'occhiata alla sequenza di eventi di esecuzione di un'istruzione. **L'esecuzione di ciascuna istruzione deve iniziare con un passo che porta il codice binario d'istruzione dalla memoria di programma al registro istruzioni; questo passo lo indichiamo come "fetch d'istruzioni".** Una volta che il codice binario d'istruzioni è nel registro istruzioni, l'unità di controllo si blocca, dopo essere stata opportunamente triggerata dal registro istruzioni. L'unità di controllo prende tutto il tempo che le occorre per generare gli eventi logici richiesti dall'istruzione. Dopo che tutti gli eventi logici sono stati completati, l'unità di controllo inizia il successivo fetch d'istruzioni. **In tal modo, l'esecuzione di ciascuna istruzione può essere suddivisa in una fase "fetch d'istruzione" e in una fase "fetch d'esecuzione", secondo quanto qui illustrato.**

La fase — fetch d'istruzione — impiegherà lo stesso tempo per ciascuna istruzione; mentre la fase — esecuzione d'istruzione — richiederà tempi diversi per differenti istruzioni. Alcuni microprocessori hanno tempi variabili di esecuzione d'istruzioni; ma la maggior parte, allo scopo di semplificare la logica, usano uno, o comunque molto pochi, tempi di esecuzione. **Supponiamo, per esempio, che diverse istruzioni richiedono fasi di esecuzione che durino uno, due, tre o quattro impulsi.** Se la fase del fetch d'istruzioni dura un impulso di clock, allora è possibile progettare il microprocessore in modo da eseguire le istruzioni in due, tre, quattro o cinque impulsi.

Data questa situazione, un progettista di microprocessori può decidere di far durare tutte le istruzioni cinque impulsi di clock, sprecando gli impulsi inutilizzati nel caso di istruzioni più brevi.
Se, d'altra parte, ci fossero pochissime istruzioni lunghe, **il suddetto progettista può decidere di standardizzare un tempo d'istruzione pari a quattro impulsi di clock, usando un doppio tempo per le poche istruzioni a cinque impulsi.** Ciò si può illustrare come segue:

Le istruzioni a cinque impulsi di clock usano otto impulsi come multipli di quattro

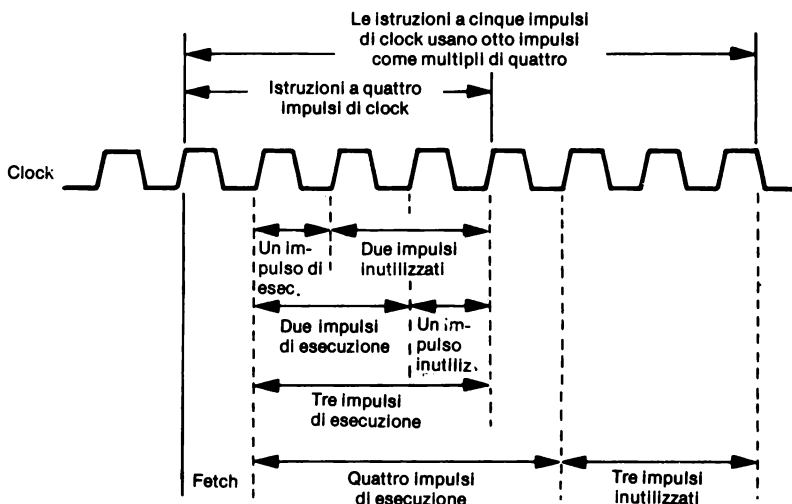
Istruzioni a quattro impulsi di clock

Clock

Un impulso di esec. | Due impulsi inutilizzati

Due impulsi di esecuzione | Un impulso inutiliz.

Tre impulsi di esecuzione

Fetch | Quattro impulsi di esecuzione | Tre impulsi inutilizzati

CICLO DI MACCHINA

In questa figura, quattro impulsi di clock costituiscono un importante intervallo di tempo per l'unità di controllo; si tratta cioè dell'unità di tempo presa come base per l'esecuzione di tutte le istruzioni. Tale unità di tempo è spesso indicata come ciclo di macchina.
Non tutti i microprocessori usano un ciclo di macchina ad intervallo di tempo fisso per dare la cadenza all'esecuzione delle istruzioni; ma per quelli che lo adottano, tutte le istruzioni saranno eseguite in un qualche numero fisso di cicli di macchina (normalmente uno, ma a volte anche due o tre).

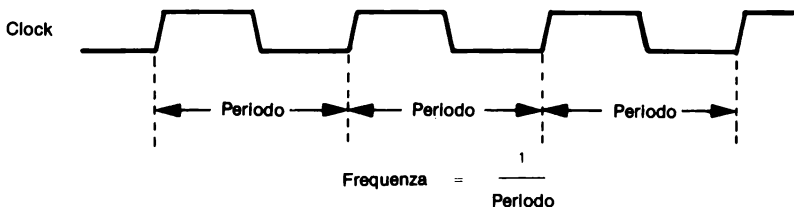
In ogni caso tutte le istruzioni sono eseguite secondo un qualche numero fisso di impulsi di clock. **Perciò la velocità con la quale il vostro microcomputer esegue i programmi varierà linearmente con la velocità del segnale di clock.**

FREQUENZA DEL SEGNALE DI CLOCK

La velocità di un segnale di clock va anche riferita al valore della sua frequenza. Le relazioni fra queste grandezze sono indicate in figura:

Clock

Periodo | Periodo | Periodo

$$\text{Frequenza} = \frac{1}{\text{Periodo}}$$

PERIODO DI CLOCK

NANOSECONDI

MICROSECONDI

Il periodo di clock è il tempo che intercorre fra due identiche forme d'onda ricorrenti. Normalmente i periodi di clock si misurano in nanosecondi. Un nanosecondo è uguale ad un millesimo di milionesimo di secondo (1×10^{-9} secondi). **Valori tipici del periodo di un segnale di clock sono compresi fra cento e cinquecento nanosecondi.** Alcuni microprocessori arrivano anche ad usare un segnale di clock di un microsecondo, uguale cioè ad un millionesimo di secondo.

MEGAHERTZ

MHz

La frequenza di un segnale è il reciproco del periodo; si può cioè dire che essa è uguale ad un secondo diviso per il periodo. Quindi, se il periodo è un microsecondo, la frequenza sarà di un milione di impulsi al secondo. **Un segnale costituito da un milione di impulsi al secondo si dice che ha frequenza di un megahertz (MHz).**

Un segnale con un periodo di 500 nanosecondi avrà frequenza pari a due milioni di impulsi al secondo; infatti:

$$\frac{1}{500 \times 10^{-9}} = \frac{1}{5 \times 10^{-7}} = \frac{10.000.000}{5} = 2.000.000$$

Perciò questo segnale avrà frequenza di due megahertz (2 MHz). Un segnale con periodi di cento nanosecondi sarà costituito da dieci milioni di impulsi al secondo:

$$\frac{1}{100 \times 10^{-9}} = \frac{1}{1 \times 10^{-7}} = 10.000.000$$

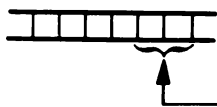
Si dirà perciò che tale segnale ha frequenza di dieci megahertz (10 MHz).

ACCESSO ALLA MEMORIA

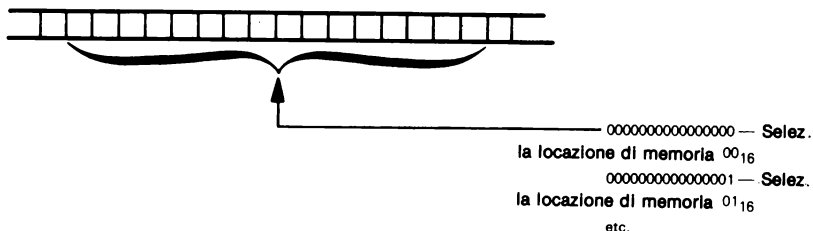
Entro un sistema microcomputer non può succedere granché prima che si verifichi l'accesso alla memoria.

Per esempio, qualsiasi istruzione inizia con un fetch d'istruzioni, che richiede che il contenuto di qualche locazione identificabile di memoria di programma sia sottoposta a fetch, come dati, e caricata nel registro istruzioni.

Similmente, ogni dato che sia in un registro dati deve essere sottoposto a fetch sia da una unità fisica esterna sia dalla memoria dati; i risultati delle operazioni della ALU nei registri dati devono essere rimandati alla memoria dati.

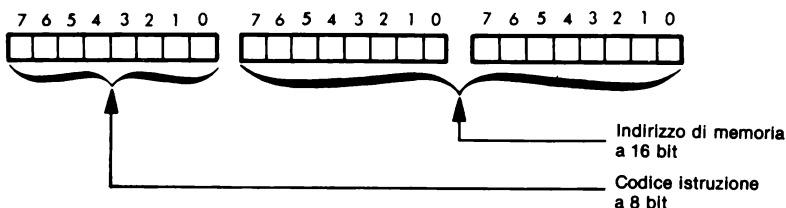
Poiché perfino una piccola memoria possiede più di mille locazioni indirizzabili, il metodo per identificare la singola locazione alla quale si desidera accedere non è di così immediata comprensione.

00 = registro dati A
01 = registro dati B
10 = registro dati C
11 = registro dati D

E' stato poco fa indicato come sia possibile identificare uno dei quattro registri dati, usando due bit del codice istruzioni. Questo semplice metodo di indirizzare i registri dati, se applicato ad una memoria esterna, può arrivare ad indirizzare fino a 65.536 locazioni di memoria; per fare questo, occorre un numero a 16 digit binari.

0000000000000000 — Selez. la locazione di memoria 00_{16}
0000000000000001 — Selez. la locazione di memoria 01_{16}
etc.

Ora, molti microcomputer vi permettono sì di indirizzare direttamente una locazione di memoria fornendo un indirizzo di memoria a 16 bit; ma ciò significa che l'istruzione è rappresentata da tre byte di dati, non da uno:

7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0

Indirizzo di memoria a 16 bit

Codice istruzione a 8 bit

Ci sono comunque alcuni problemi associati con questo modo di generare indirizzi di memoria. In primo luogo, esso comporta **un certo spreco di memoria.** Data la frequenza con la quale si susseguono le istruzioni, ed il fatto che la maggior parte dei microcomputer non hanno qualcosa come 65.536 byte di memoria effettivamente disponibili, è senz'altro augurabile una qualche procedura di indirizzamento memoria alternativa, e più economica.

Inoltre, nella maggioranza dei sistemi microcomputer, **sebbene si parli di "memoria di programma" e di "memoria dati", essi sono in genere la stessa cosa:**

Questa parte di memoria contiene il programma

questa parte di memoria contiene i dati

Quando si scrive un programma per un microcomputer, si decide quale parte di memoria diventi memoria di programma; il resto sarà la memoria dati. Ma la volta successiva, scrivendo un programma per lo stesso microcomputer, si potrà trovare

memoria localizzata fra le aree di programma e di dati in modi completamente diversi.

Memoria di programma

Memoria di programma

Memoria di progr

Memoria di programma

Capita spesso di scrivere programmi semplici (per esempio, per effettuare operazioni aritmetiche su numeri che indichino somme in dollari); **i programmi risulteranno molto più utili se voi potrete riutilizzarli in molte applicazioni.**

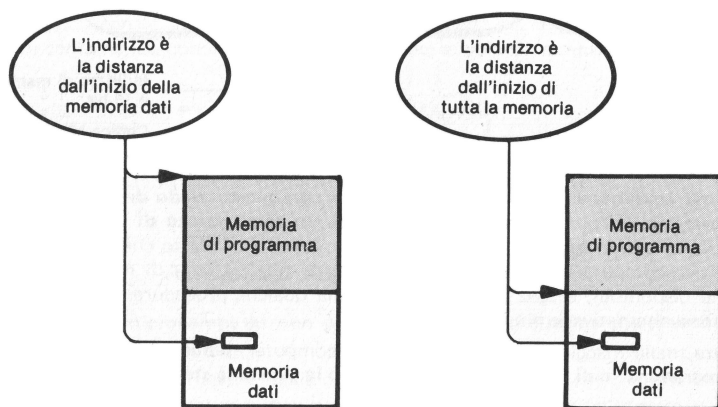
Per fare questo, occorrerà una qualche tecnica di indirizzamento memoria che identifichi una locazione di memoria dati in termini di dislocazione (o distanza) rispetto all'inizio dell'area di memoria dati, piuttosto che rispetto all'inizio della memoria:

L'indirizzo è la distanza dall'inizio della memoria dati

Memoria di programma

Memoria dati

L'indirizzo è la distanza dall'inizio di tutta la memoria

Memoria di programma

Memoria dati

Esaminiamo i vantaggi che derivano dall'identificare una locazione di memoria in base alla sua dislocazione rispetto all'inizio dell'area dati.

Il vantaggio più importante consiste nella possibilità di muovere tale area dati senza dover cambiare i programmi che danno accesso alla stessa.

Per comprendere come ciò avviene, consideriamo il problema quotidiano contrassegnando il passare del tempo.

Il tempo può essere identificato dall'ora del giorno — che equivale all'indirizzamento assoluto della memoria —, oppure riferendoci ad un qualche numero di ore che precedono o seguono un certo evento; ciò equivale ad indirizzare la memoria come dislocazione dall'inizio dell'area dati. Supponiamo, per esempio, di dover lasciare le istruzioni per la preparazione del pranzo; tali istruzioni possono essere del tipo:

1) accendere il forno alle 6 p.m.;
2) alle 6.15 mette il pollo nel forno;
3) alle 8.15 togliere il pollo;
4) alle 8.30 servire in tavola.

Se si cambia l'ora del pranzo, occorre cambiare tutto il set di istruzioni per la sua preparazione. In altro caso, si possono riscrivere le suddette istruzioni come segue:

1) accendere il forno 2 ore e mezza prima del pranzo;
2) a 2 ore e un quarto prima del pranzo mettere il pollo nel forno;
3) 15 minuti prima del pranzo togliere il pollo dal forno;
4) servire in tavola all'ora di pranzo.

Poiché queste istruzioni riferiscono il tempo alla distanza rispetto all'ora di pranzo cambiando quest'ultime non cambiano le istruzioni. Chiaramente, questo è un sistema molto più utile per programmare il tempo; ciò è quello che equivale ad indirizzare la memoria mediante la sua dislocazione rispetto all'inizio dell'area dati.

Senza esplorare ulteriormente le idee da cui nascono gli indirizzi di memoria, è chiaro che basta indicare un caso per avere una varietà di differenti tecniche creative degli indirizzi di memoria.

INDIRIZZAMENTO DI MEMORIA

L'indirizzamento della memoria è il termine usato per descrivere in generale le tecniche che un microprocessore permette di usare allo scopo di identificare le locazioni di memoria. Vediamo con esattezza come lavorano gli indirizzi di memoria.

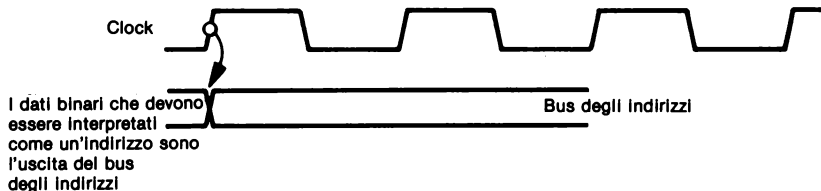
LOGICA DI INDIRIZZAMENTO DI MEMORIA

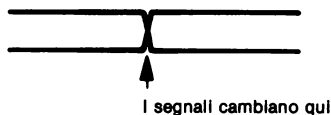
La memoria sostanzialmente consiste di dispositivi logici che gestiscono tre tipi di informazioni: indirizzi, dati e segnali di controllo.

Ciascun accesso di memoria parte con un indirizzo da trasmettere al dispositivo di memoria:

Clock

I dati binari che devono essere interpretati come un'indirizzo sono l'uscita del bus degli indirizzi

Bus degli indirizzi

Dalla discussione fatta più indietro in questo capitolo, ricorderete che avevamo usato il simbolo:

I segnali cambiano qui

per identificare l'istante in cui uno o più segnali su un bus a più linee possono mutare il loro livello. Per un singolo segnale, possiamo indicare in concreto la variazione, da basso ad alto:

o da alto a basso

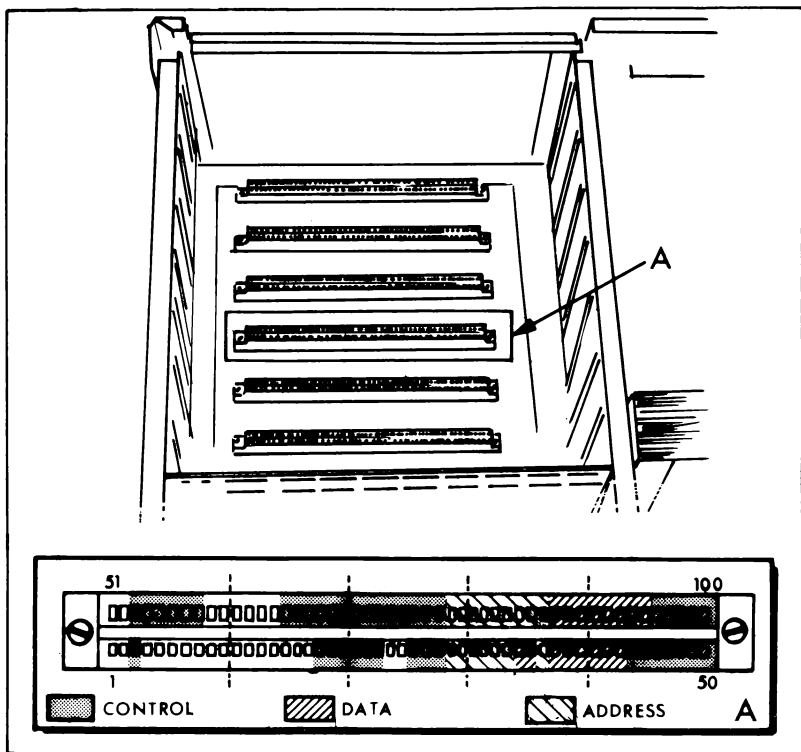



Figura 6-2. I bus di un sistema microcomputer

BUS DEGLI INDIRIZZI

Un indirizzo di memoria consiste in dati binari che vengono inviati in uscita su un bus appropriato che chiamiamo (per ragioni evidenti) **bus degli indirizzi.** Questo, come qualsiasi altro bus, è un certo numero di conduttori paralleli che collegano la CPU ai dispositivi di memoria del sistema microcomputer, come illustrato in fig. 6-2.

BUS DEI DATI

Ci sarà bus dei dati separato, che collega la CPU ai dispositivi di memoria; pure esso è illustrato in fig. 6-2. Una volta che la CPU ha inserito dati binari su un bus, non esiste più alcuna possibilità di errore di interpretazione. Il dato binario che compare sul bus degli indirizzi deve essere interpretato come un indirizzo, mentre il dato binario che compare sul bus dei dati deve essere interpretato come un dato.

Le illustrazioni temporali di questo capitolo mostrano gli indirizzi che compaiono sul bus degli indirizzi in corrispondenza del fronte di salita dell'impulso di clock.
La logica entro la CPU userà il fronte di salita di un appropriato impulso di clock come trigger per creare segnali di controllo che interconnettano le linee del bus degli indirizzi con il registro dei dati entro la CPU; è questa connessione che fa si che il dato divenga un indirizzo.

La logica entro la CPU sarà probabilmente in grado di collegare i bus dei dati e degli indirizzi agli stessi registri dati.

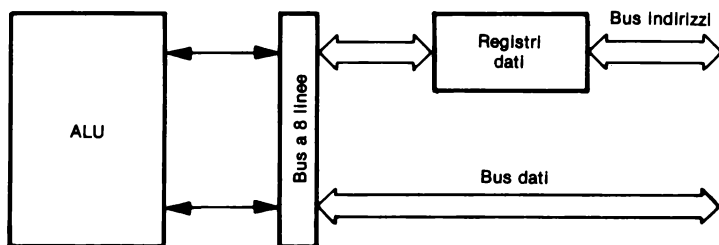
Ciò consente di calcolare gli indirizzi come qualunque altro dato, prima di porre i dati in uscita come indirizzi nel bus degli indirizzi; ciò si può illustrare come segue:

ALU

Bus a 8 linee

Registri dati

Bus indirizzi

Bus dati

Il punto concettuale più importante per capire ciò è che la logica entro la CPU può interpretare i dati binari in qualsiasi maniera. Ma una volta che i dati appaiono su un bus al di fuori della CPU, tale bus determinerà, entro un certo limite, come tali dati vadano interpretati. Per esempio, l'informazione su un bus degli indirizzi deve essere un indirizzo di memoria; esso però può provenire da qualsiasi zona della CPU, e, per quanto concerne la logica esterna, si suppone solo che essa interpreti un indirizzo. Se la logica esterna usa i bus degli indirizzi in qualche altro modo, la CPU non interferisce, ma... la logica dovrebbe meglio conoscere cosa sta facendo.

Guardiamo ora il modo in cui la CPU e i dispositivi di memoria conversano attraverso le linee dei bus.

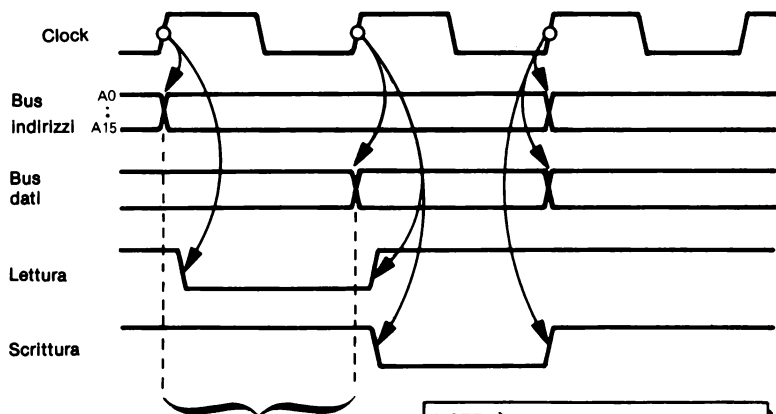
Ciascuna locazione di memoria entro l'apposito dispositivo avrà un unico indirizzo associato con essa. Il dispositivo di memoria decodifica il dato binario sul bus indirizzi e lo usa per "selezionare" una singola locazione di memoria. Susseguentemente, il dato può essere trasmesso al dispositivo di memoria, tramite il bus dati, nel qual caso, per un'operazione di scrittura, esso verrà immagazzinato nella locazione di memoria prescelta; oppure, per una operazione di lettura, il dato può essere trasmesso dal dispositivo di memoria, tramite il bus dati, alla CPU. Segnali di controllo generati dalla CPU determinano se essa "legge" dati dalla memoria o "scrive" dati in memoria. **Cosicché, l'interconnessione fra memoria e CPU consisterà di un bus indirizzi, un bus dati e segnali di controllo; tutto ciò è illustrato in fig. 6-2 e rappresentato sotto l'aspetto funzionale come segue:**

Clock

Bus indirizzi A0 : A15

Bus dati

Lettura

Scrittura

Entro questo tempo il dispositivo di memoria decodifica il bus degli indirizzi e seleziona una locazione di memoria.

NOTE: È indicato un segnale di trigger mentre sta provocando lo scatto in più di uno dei livelli di segnale triggerati

Incorporati in questa figura ci sono alcuni concetti importanti. Consideriamo prima di tutto il bus degli indirizzi; esso è indicato come costituito di 16 linee parallele, che è il numero più comune per il bus indirizzi dei normali microcomputer.

Un bus indirizzi a 16 linee può trasportare un indirizzo a 16 bit, capace quindi di indirizzare 65.536 diverse locazioni di memoria.

L'indirizzo stesso è indicato sotto azione di gate sul fronte di salita di un impulso di clock. L'indirizzo è mantenuto sul bus per 2 impulsi di clock; per questo periodo di tempo la CPU mantiene le linee del bus indirizzi collegate ai bit del registro dati, fuori dal quale si genera l'indirizzo.

Ci riferiamo a questo periodo di tempo come il periodo durante il quale l'indirizzo è "stabile" sul relativo bus.

Esaminiamo ora i bus dati.

Ancora una volta ci sarà qualche periodo di tempo durante il quale il dato sul bus dati deve essere stabile; e questo periodo sarà ancora definito dai fronti del segnale di clock. Il bus dei dati inizierà ad essere stabile qualche istante dopo che lo è il bus degli indirizzi; ciò è necessario in quanto si deve dar tempo alla logica del dispositivo di memoria di ricevere e rispondere all'indirizzo, cosicché, nel momento in cui il dato è stabile, c'è tempo di selezionare la locazione di memoria indirizzata.

CONTROLLO DI LETTURA IN MEMORIA

Se sta per verificarsi un'operazione di lettura in memoria, il segnale di controllo di lettura arriverà sotto forma d'impulso. Tale impulso viene arbitrariamente illustrato come basso, ma potrebbe con la stessa facilità essere esemplificato come alto. In ogni caso, ciò che è importante è che il segnale di controllo presenta uno stato "passivo" durante il quale non succede niente, ed uno stato "attivo" durante il quale esso indica che qualcosa sta succedendo; in tal caso, si tratta di una operazione di lettura.

Un'operazione di lettura richiede che il contenuto della locazione di memoria in indirizzo sia inserito nel bus dati. Ancora una volta, serve un po' di tempo affinché la logica del dispositivo di memoria avverta l'impulso basso di lettura e risponda ponendo i dati dalla locazione di memoria al bus relativo.

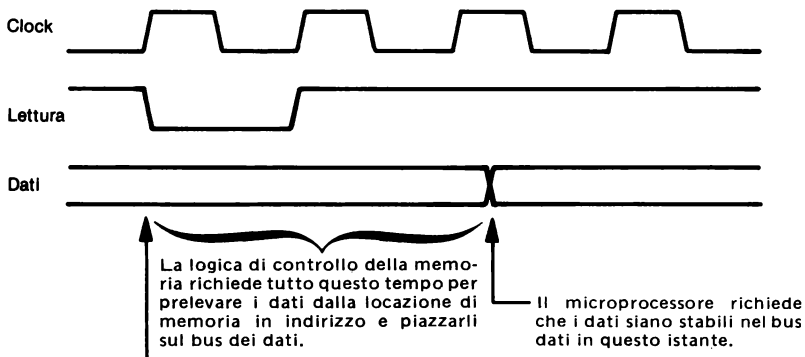
L'impulso basso di lettura può quindi verificarsi abbastanza presto da dare alla logica di memoria il tempo per trasferire i dati dalla locazione di memoria prescelta al bus dati:

Clock

Lettura

Dati

La logica di controllo della memoria richiede tutto questo tempo per prelevare i dati dalla locazione di memoria in indirizzo e piazzarli sul bus dei dati.

Il microprocessore richiede che i dati siano stabili nel bus dati in questo istante.

Perciò il segnale di controllo lettura qui deve andar basso

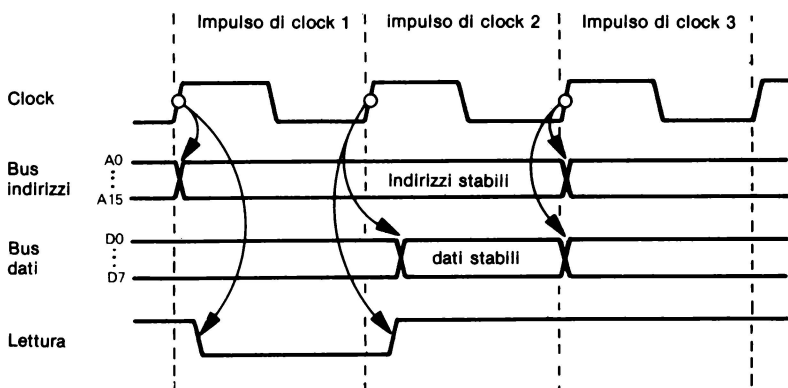
CONTROLLO DI SCRITTURA IN MEMORIA

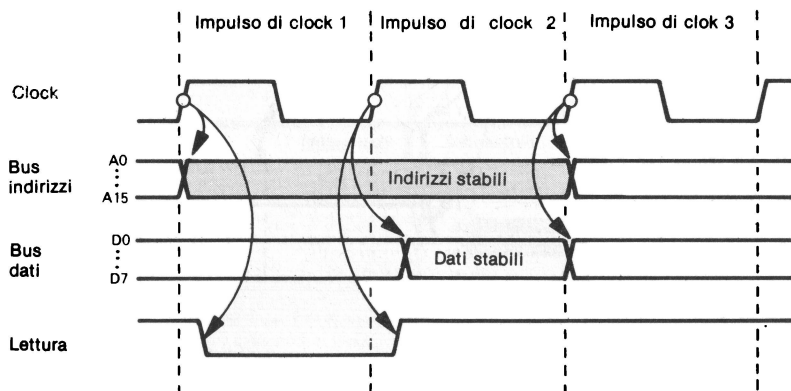
Il segnale di controllo di scrittura, che è ancora indicato come un segnale impulsivo basso, indica che il dato sul bus dati deve essere scritto nella parola di memoria indirizzata. Il segnale di controllo scrittura agisce come uno strobe; esso non può verificarsi come "vero" (un impulso basso è "vero") finchè un dato valido sia stabile sul bus dati.

Ciò avviene in quanto il dispositivo di memoria userà lo strobe basso di scrittura per connettere il bus dati alla locazione di memoria indirizzata. Ove lo strobe basso di scrittura si verifichi prima che i dati siano stabili sul bus, dati erronei possono venir scritti nel dispositivo di memoria.

OPERAZIONE DI LETTURA IN MEMORIA

Esaminiamo ora le operazioni di lettura e scrittura in memoria individualmente. Prima di tutto c'è una temporizzazione associata con l'operazione di lettura:

Impulso di clock 1
Impulso di clock 2
Impulso di clock 3
Clock
Bus indirizzi
A0
A15
Indirizzi stabili
Bus dati
D0
D7
dati stabili
Lettura

L'operazione di lettura in frequenza inizia sul fronte di salita dell'impulso di clock 1; la CPU usa quest'impulso per collegare gli appropriati registri dati al bus indirizzi e ciò provoca la comparsa di un indirizzo stabile sul bus:

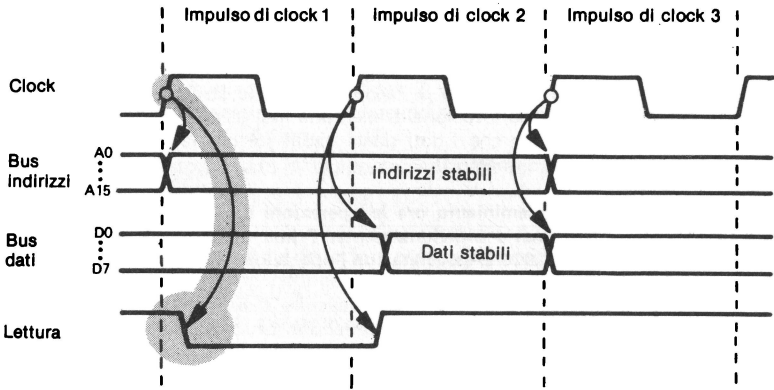
Impulso di clock 1
Impulso di clock 2
Impulso di clok 3
Clock
Bus indirizzi
A0
A15
Indirizzi stabili
Bus dati
D0
D7
Dati stabili
Lettura

Dopo pochi istanti, il segnale di controllo lettura va basso:

Impulso di clock 1 | Impulso di clock 2 | Impulso di clock 3

Clock

Bus indirizzi A0 ... A15 — Indirizzi stabili

Bus dati D0 ... D7 — Dati stabili

Lettura

La logica della memoria decodifica il contenuto del bus indirizzi per identificare una singola locazione di memoria come "selezionata".

La logica di questa locazione scelta non è altro che una combinazione di operazioni Booleane effettuate sui singoli segnali del bus indirizzi, comunque per ora il metodo effettivo usato da un dispositivo di memoria per decodificare i contenuti del bus indirizzi non è importante.

Tutto ciò che importa è recepire che esistono 65.536 possibilità di scegliere una singola locazione di memoria. Il segnale di controllo lettura fa sì che il dispositivo di memoria connetta i bit della locazione di memoria al bus dei dati.

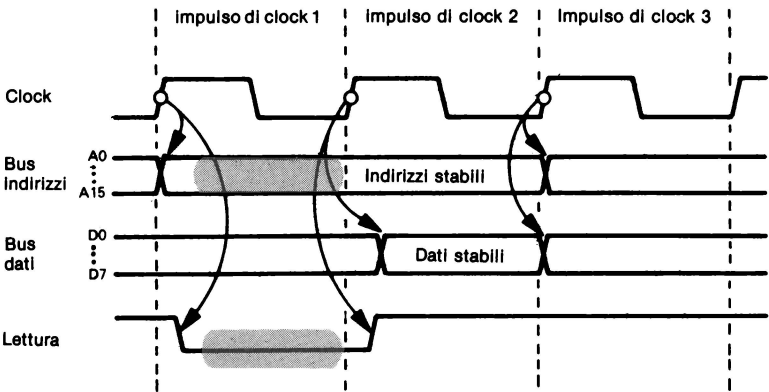
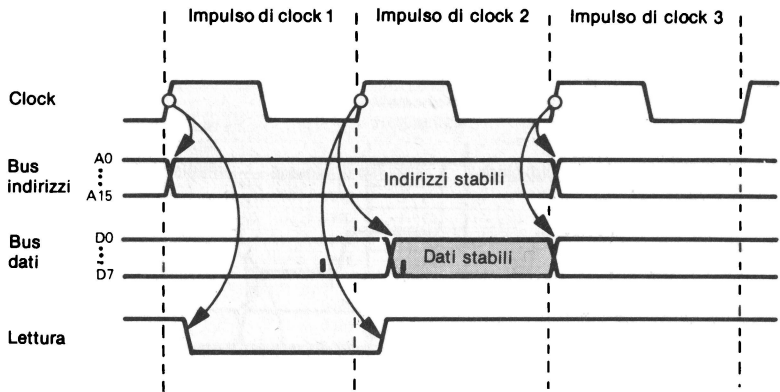
Il processo di identificazione di una singola locazione di memoria, e di connetterla poi con un bus dati, può verificarsi prima del fronte d'inizio dell'impulso di clock 2:

Impulso di clock 1 | Impulso di clock 2 | Impulso di clock 3

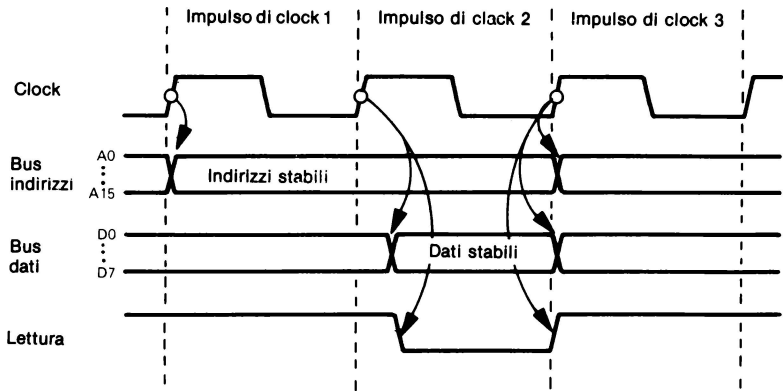
Clock

Bus indirizzi A0 ... A15 — Indirizzi stabili

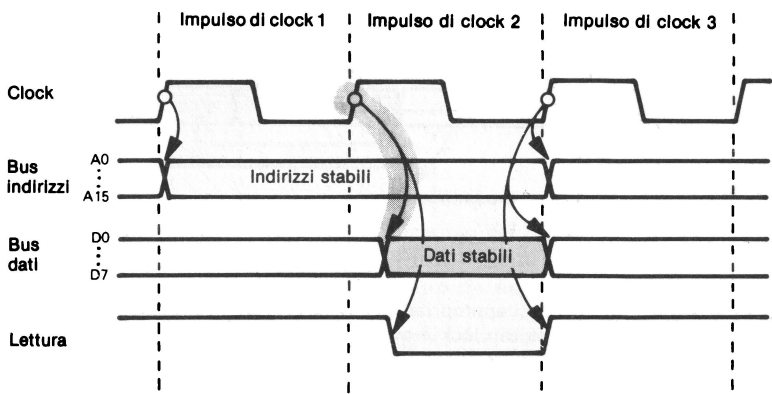
Bus dati D0 ... D7 — Dati stabili

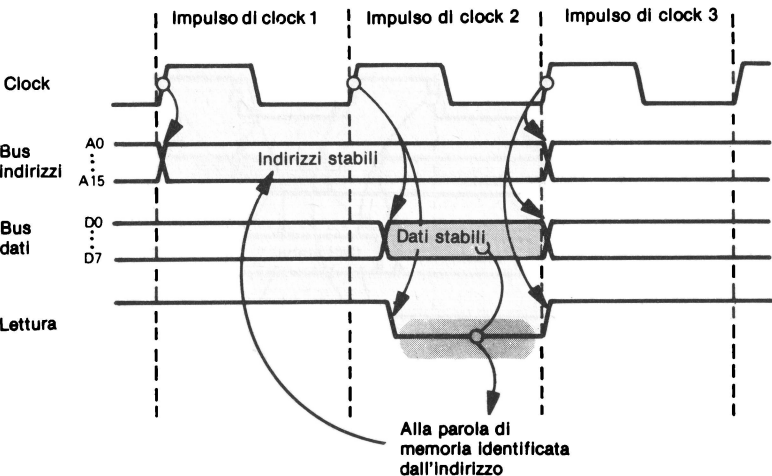
Lettura

A questo punto, il contenuto della locazione di memoria selezionata deve essere stabile sul bus dei dati. Questi dati devono essere mantenuti stabili per un certo tempo finito e definito; questo è il tempo che deve essere dato alla CPU allo scopo di estrarre i dati dal bus dati e caricarli in un opportuno registro:

La CPU collega il registro dati appropriati, e così la lettura è completa. Nell'illustrazione qui sopra, un impulso di clock è evidenziato come il tempo durante il quale i dati sul bus dati sono mantenuti stabili.

OPERAZIONE DI SCRITTURA IN MEMORIA

Consideriamo ora un'operazione di scrittura in memoria; essa si può illustrare come segue:

La temporizzazione associata con un'operazione di scrittura in memoria non differisce in modo sostanziale da quella associata con un'operazione di lettura da memoria. La logica che fornisce in uscita un indirizzo stabile sul bus indirizzi è di fatto identica per le due operazioni. Differenze esistono sul bus dati e sui segnali di controllo. Quando l'indirizzo compare sul bus indirizzi non c'è ad accompagnarlo alcun segnale basso di controllo scrittura; perciò la locazione di memoria scelta non risulta collegata al bus dati ed il suo contenuto non è fornito in uscita sul bus dati.

Di conseguenza la CPU, allo scopo di fornire dati stabili in uscita, collega al bus un appropriato registro dati:

Impulso di clock 1
Impulso di clock 2
Impulso di clock 3
Clock
Bus indirizzi
A0
A15
Indirizzi stabili
Bus dati
D0
D7
Dati stabili
Lettura

A questo punto, il segnale di controllo scrittura è triggerato basso; esso verrà ricevuto dal dispositivo di memoria e usato per collegare la locazione di memoria prescelta al bus dati, cosicché il contenuto del bus dati viene scritto nella locazione scelta e memorizzato permanentemente:

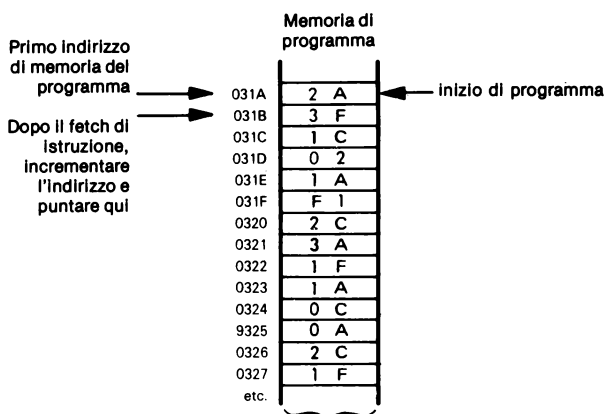
Impulso di clock 1
Impulso di clock 2
Impulso di clock 3
Clock
Bus indirizzi
A0
A15
Indirizzi stabili
Bus dati
D0
D7
Dati stabili
Lettura
Alla parola di memoria identificata dall'indirizzo

INDIRIZZAMENTO DELLA MEMORIA DI PROGRAMMA E CONTATORE DI PROGRAMMA

Esaminiamo ora i registri della CPU che sono richiesti per generare gli indirizzi di memoria; cominciamo con lo studiare l'indirizzamento della memoria di programma.

La logica richiesta per indirizzare la memoria di programma è assolutamente "lineare". Un programma non è altro che una sequenza di codici di istruzioni che sono normalmente immagazzinati nelle locazioni di memoria, incrementandone via via gli indirizzi:

Primo indirizzo di memoria del programma →

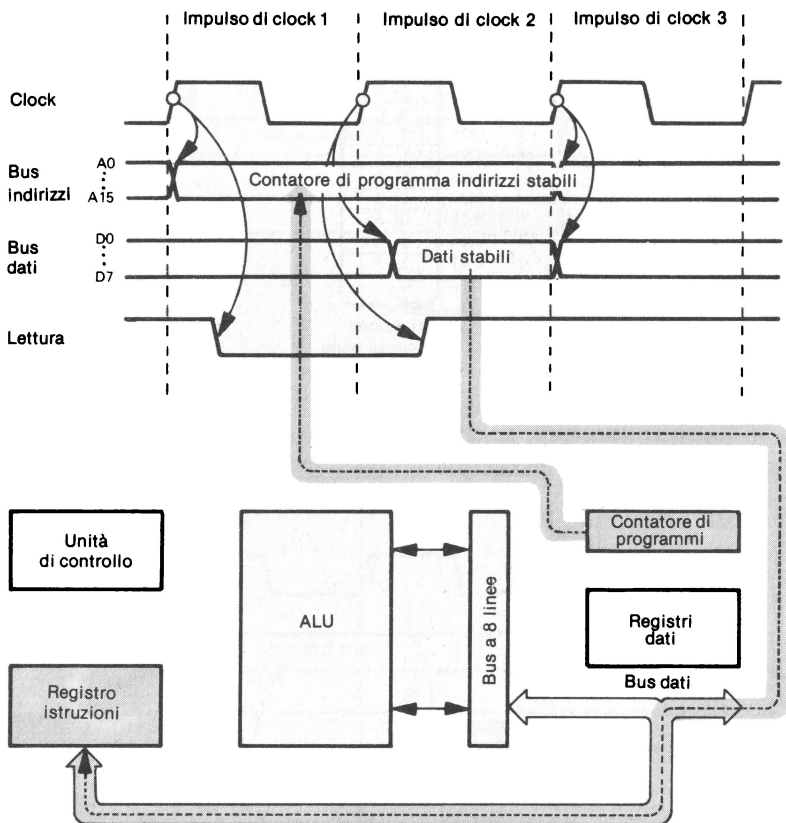
Dopo il fetch di istruzione, incrementare l'indirizzo e puntare qui →

	Memoria di programma	
031A	2 A	← inizio di programma
031B	3 F	
031C	1 C	
031D	0 2	
031E	1 A	
031F	F 1	
0320	2 C	
0321	3 A	
0322	1 F	
0323	1 A	
0324	0 C	
9325	0 A	
0326	2 C	
0327	1 F	
etc.		

Questi numeri esadecimali sono stati scelti arbitrariamente per rappresentare i codici di istruzione.

CONTATORE DI PROGRAMMA

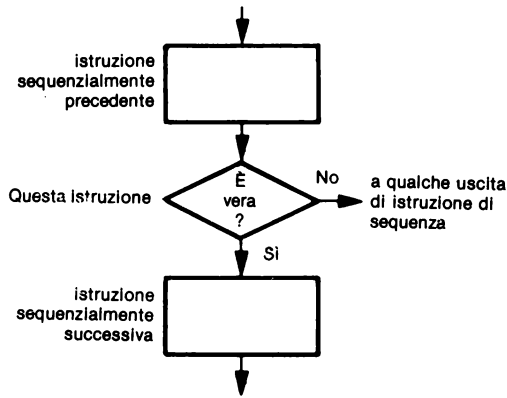
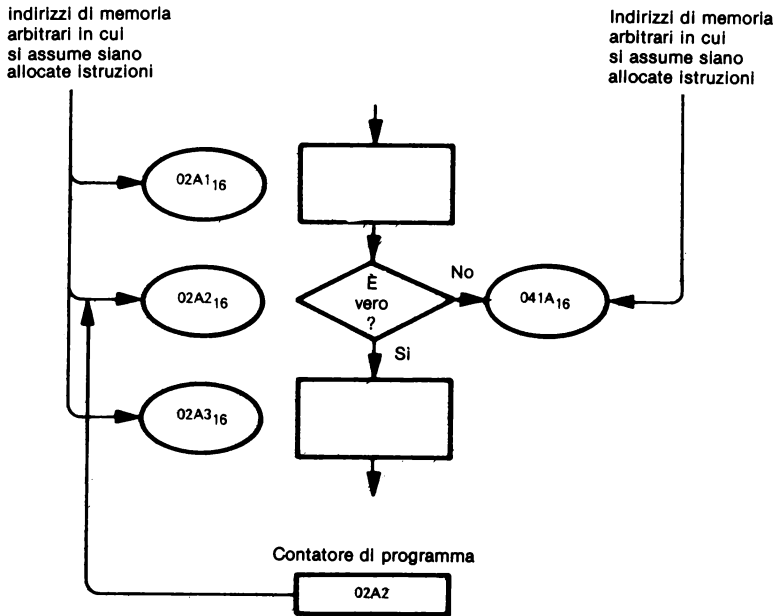
Così stando le cose, tutto ciò che dobbiamo fare è identificare l'indirizzo della prima istruzione entro la sequenza. Dopo ogni fetch d'istruzione, se si incrementa questo indirizzo esso punterà con precisione alla successiva istruzione in sequenza.

Questa logica di indirizzamento della memoria di programma è elaborata da un registro indicato come contatore di programma.

Esso contiene dati binari che vengono interpretati come un indirizzo di memoria solamente in quanto i dati binari sono portati in uscita sul bus indirizzi. Così il contatore di programma fornisce l'indirizzo di memoria per tutte le operazioni di fetch istruzioni; alla memoria esterna, il fetch istruzioni appare esattamente come una qualsiasi operazione di lettura di memoria. Tutto ciò può indicarsi come segue:

LOGICA DI PROGRAMMA E CONTATORE DI PROGRAMMA

Il contatore di programma si configura, all'interno della CPU, sia come un registro per la generazione degli indirizzi, sia come un registro dati.

E' molto importante che il contatore di programma sia accessibile come un registro dati, in quanto questa è la base per la logica di programma.

Abbiamo già visto, nei grafici di flusso, degli elementi di decisione i quali fan sì che il programma continui nell'esecuzione delle istruzioni sequenzialmente successive, o di qualche istruzione fuori sequenza:

Come possiamo identificare una istruzione fuori sequenza e quindi attuarne il fetch? La risposta sta nel conoscere in anticipo l'indirizzo di questa istruzione fuori programma. Se carichiamo questo indirizzo noto, come dati, nel contatore di programmi, eseguiamo quella che si chiama un'istruzione jump (di salto). Quando si esegue l'istruzione jump, il contatore di programma punta verso l'istruzione stessa:

Dopo che dell'istruzione jump si è attuato il fetch, si lascerà che il contatore di pro-

gramma punti all'istruzione sequenziale successiva:

Indirizzi di memoria arbitrari in cui si assume siano allocate istruzioni

Indirizzi di memoria arbitraria in cui si assume siano allocate istruzioni

$02A1_{16}$

$02A2_{16}$

$02A3_{16}$

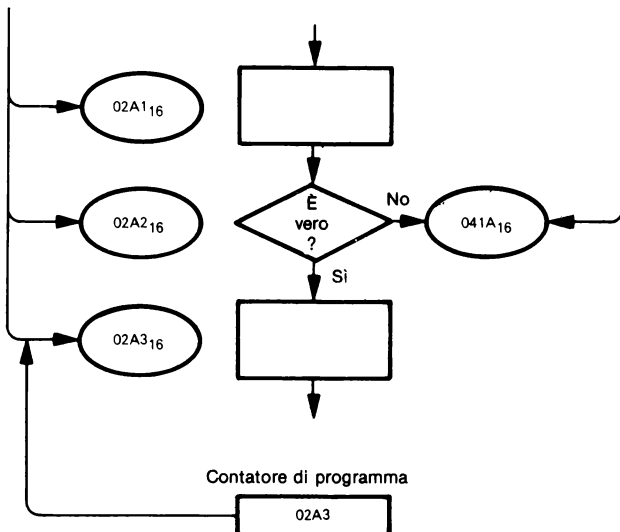
È vero ?

No

Sì

$041A_{16}$

Contatore di programma

02A3

Se sta per verificarsi il jump, allora la CPU lo provocherà caricando l'indirizzo dell'istruzione fuori sequenza, come fossero dati, nel contatore di programma; e ciò viene effettuato durante la fase di esecuzione dell'istruzione del jump:

Indirizzi di memoria arbitrari in cui si assume siano allocate istruzioni

Indirizzi di memoria arbitrari in cui si assume siano allocate istruzioni

$02A1_{16}$

$02A2_{16}$

$02A3_{16}$

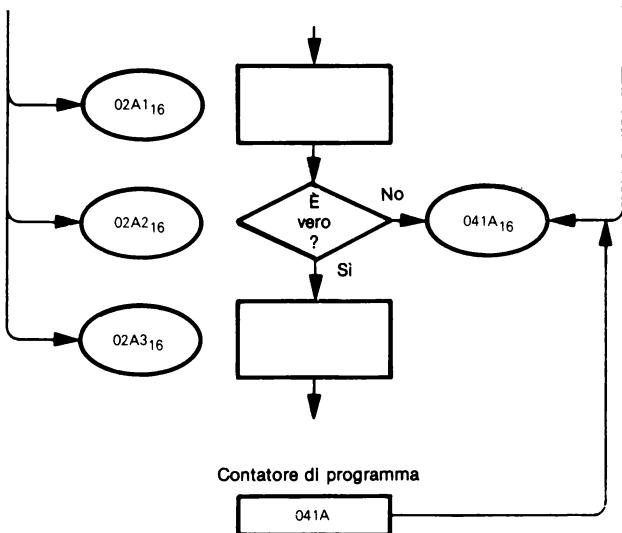
È vero ?

No

Sì

$041A_{16}$

Contatore di programma

041A

Ora, quando l'istruzione di jump ha esaurito la sua esecuzione, e parte l'esecuzione dell'istruzione successiva, si verificherà un fetch; ma, invece di prendere l'istruzione in sequenza successiva, si prende la nuova istruzione fuori sequenza.

REGISTRI DI INDIRIZZAMENTO DELLA MEMORIA DATI

Come già abbiamo visto in questo capitolo, l'indirizzamento della memoria dati non è così diretto ed immediato come quello della memoria di programma, in quanto non c'è alcuna sequenza "usuale" in cui i dati possano essere memorizzati.

Cosicché si usa una varietà di tecniche più o meno ingegnose per creare indirizzi per la memoria dati. Il computo degli indirizzi della memoria dati diventa una semplice estensione di logica nella zona dati della ALU.

Disponendo di registri dati in cui immagazzinare i dati binari, più una ALU che possiamo usare per effettuare calcoli, possediamo tutti i requisiti preliminari per calcolare dati binari che diventaranno poi indirizzi di memoria dati.

Cosicché il computo degli indirizzi di memoria dati è facilmente manipolato da logica già presente in una CPU.

In questo volume non andremo a discutere i vari metodi usati per generare gli indirizzi di memoria dati. Queste informazioni diventano necessarie solo quando si deve partire per imparare come scrivere i programmi in linguaggio assembly, e questo sarà argomento del volume 1°.

Un indirizzo di memoria dati è il risultato di computi di dati che hanno preceduto un accesso di dati; e ciò è tutto quanto serve conoscere per ora.

INDIRIZZAMENTO DELLA LOGICA ESTERNA

Allo scopo di indirizzare qualche dispositivo logico esterno o periferico (come tastiera o video display), la CPU passerà attraverso una sequenza di passi che sono molto simili a quelli richiesti per accedere alla memoria.

Il dispositivo logico esterno da indirizzare avrà un indirizzo specifico, esattamente come avviene per una qualsiasi locazione di memoria.

Quindi, l'accedere ad un dispositivo esterno comprende questi passi:

1) generare un indirizzo è un numero binario generato come dati e posto in uscita su un bus indirizzi;
2) applicare un appropriato segnale di trigger, che dice al dispositivo I/O cosa deve fare;
3) trasmettere o ricevere dati tramite un bus dati.

La sequenza di eventi associata con l'accesso della logica esterna o dei dispositivi I/O è così simile ad un riferimento di memoria che molti microprocessori trattano le due cose come fossero la stessa, unica cosa.

In tal caso, i dispositivi I/O e la logica esterna rispondono esattamente agli stessi segnali come si trattasse di un dispositivo di memoria; e la sola cosa che separa i due sono gli indirizzi di memoria che vengono messi da parte per ciascuno.

I microprocessori che trattano memoria e dispositivi I/O separatamente hanno, in effetti, un set secondario di logica che essenzialmente duplica la logica di indirizzamento di memoria, ma su scala più piccola.

Per esempio, dove il bus indirizzi per memorie normalmente consiste di 16 linee capaci di indirizzare 65.536 locazioni individuali di memoria, il bus indirizzi di I/O potrebbe essere ampio solo 8 linee, e ciò significa che possono essere indirizzati solo 256 differenti dispositivi I/O.

SET DI ISTRUZIONI E PROGRAMMAZIONE

Le istruzioni di un microprocessore identificano, come ricorderete, le operazioni individuali che possono essere effettuate come entità singole dalla logica interna del microprocessore. Ci sono cinque tipi di eventi che possono essere specificati da istruzioni individuali, e sono indicati in figura:

Istruzioni microprocessore

① Trasferimento dati tra il microprocessore e la logica esterna

CPU
DATI

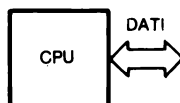
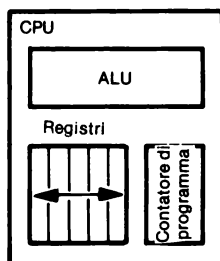
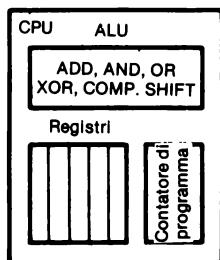
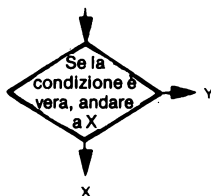
② Movimento dati da un registro ad un altro entro il microprocessore

CPU
ALU
Registri
Contatore di programma

③ Specificare se si tratta di una operazione logica o aritmetica.

CPU
ALU
ADD, AND, OR
XOR, COMP. SHIFT
Registri
Contatore di programma

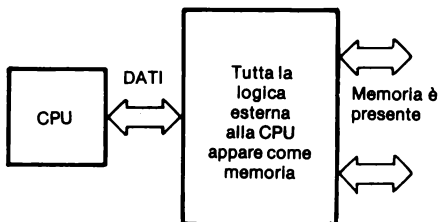
④ Modifica dei contenuti del contatore di programma per abilitare la logica programmata.

Se la condizione è vera, andare a X
Y
X

⑤ La condizione di manipolazione di status flag nell'istruzione di tipo ④ è un esempio di stato.

Consideriamo questi tipi di istruzioni una per volta.

Le istruzioni che muovono dati fra il microprocessore e la logica al di là dello stesso possono accedere alla memoria o a dispositivi fisici. In genere si indicano come dispositivi fisici al di là del microcomputer i dispositivi I/O; le istruzioni che danno accesso ad essi si chiamano istruzioni I/O. **Alcuni microprocessori presentano istruzioni separate per trasferire dati fra il microprocessore e i dispositivi I/O o la memoria:**

Istruzioni microprocessore

① Trasferimento dati tra il microprocessore e la logica esterna

CPU
DATI
Tutta la logica esterna alla CPU appare come memoria
Memoria è presente
Alcuni dispositivi I/O simulano di essere una memoria che in realtà non è presente.

② Movimento dati da un registro ad un altro entro il microprocessore

CPU
ALU
Registri
Contatore di programma

③ Specificare se si tratta di una operazione logica o aritmetica.

CPU ALU
ADD, AND, OR
XOR, COMP. SHIFT
Registri
Contatore di programma

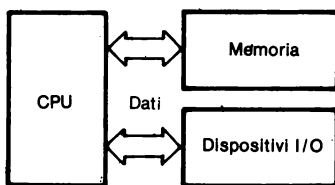
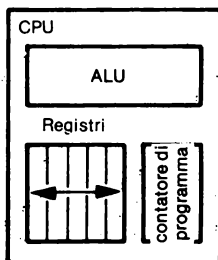
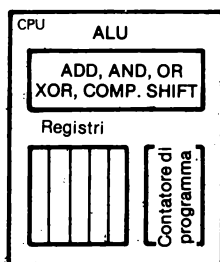
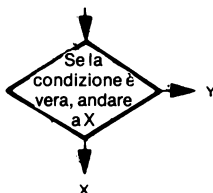
④ Modifica dei contenuti del contatore di programma per abilitare la logica programmata.

Se la condizione è vera, andare a X
Y
X

⑤ La condizione di manipolazione di status flag nell'istruzione di tipo ④ è un esempio di stato.

Altri microprocessori usano le stesse istruzioni per trasferire dati fra il microprocessore e la memoria o i dispositivi I/O:

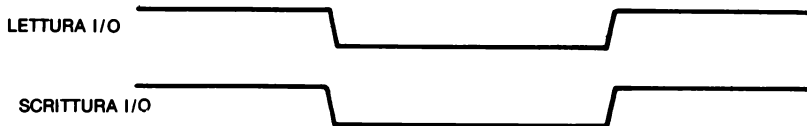
Istruzioni microprocessore

① Trasferimento dati tra il microprocessore e la logica esterna

② Movimento dati da un registro ad un altro entro il microprocessore

③ Specificare se si tratta di una operazione logica o aritmetica.

④ Modifica del contenuto del contatore di programma per abilitare la logica programmata.

⑤ La condizione di manipolazione di status flag nell'istruzione di tipo ④ è un esempio di stato.

Si ricorderà, da precedenti discussioni, sui segnali e sui bus, che **esiste veramente poca differenza fra le istruzioni relative all'I/O e quelle di riferimento in memoria.** Le istru-

zioni I/O generano segnali di controllo che identificano il trasferimento dati verso o da un dispositivo I/O:

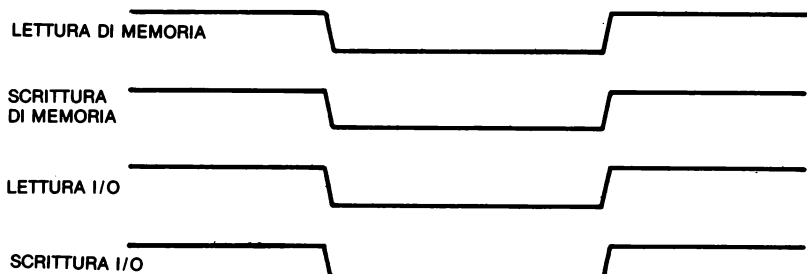
LETTURA I/O

SCRITTURA I/O

Le istruzioni di riferimento in memoria generano segnali equivalenti:

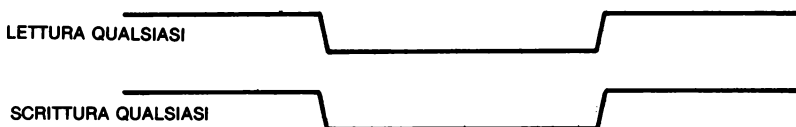
LETTURA DI MEMORIA

SCRITTURA DI MEMORIA

LETTURA I/O

SCRITTURA I/O

Un microprocessore che usi le medesime istruzioni per accedere alla memoria o ad un I/O avrà semplicemente un set di segnali di controllo:

LETTURA QUALSIASI

SCRITTURA QUALSIASI

Quando un microprocessore usa gli stessi segnali di controllo per accedere alla memoria o ai dispositivi I/O, la logica associata con l'indirizzo di memoria distingue fra memoria o dispositivo di I/O.

Chiunque progetti il microcomputer decide quale sarà l'indirizzo ad accedere in effetti alla memoria, e quale indirizzo provvederà invece a selezionare l'I/O.

Da notare che, proprio perché un microprocessore ha istruzioni di memoria e di I/O separate, un progettista di microcomputer non deve usare le istruzioni I/O.

Il progettista potrebbe anche indirizzare i dispositivi I/O come se si trattasse di locazioni di memoria. Non è vero l'inverso; un microprocessore che non ha istruzioni I/O costringe il progettista a indirizzare i dispositivi I/O come locazioni di memoria.

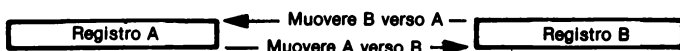
In ogni caso, l'utente non ha scelta: si devono usare i riferimenti di memoria e le istruzioni I/O esattamente come indicato dal progettista del microcomputer.

Il numero e la complessità delle istruzioni che spostano i dati fra i registri della CPU sono strettamente funzione dei registri propri del microprocessore.

Ovviamente, un microprocessore che abbia un solo registro indirizzabile non avrà alcuna istruzione da spostare fra registri.

Un microprocessore che abbia due registri, potrebbe avere due istruzioni di questo tipo:

Registro A ← Muovere B verso A — Registro B

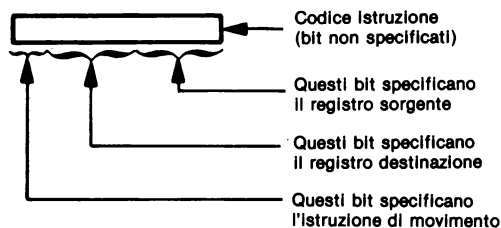
Registro A — Muovere A verso B → Registro B

Come il numero dei registri interni alla CPU aumenta, il microprocessore arriverà ad un certo punto oltre le proprie possibilità.

Supponiamo, per esempio, che un microprocessore abbia 16 registri indirizzabili; per muovere i dati da ognuno di questi 16 registri (come partenza) a ciascuno dei 16 registri (come destinazione), il codice istruzioni del micro deve avere alcuni bit per identificare ciascun possibile registro di partenza e ciascun possibile registro d'arrivo:

Codice istruzione (bit non specificati)

Questi bit specificano il registro sorgente

Questi bit specificano il registro destinazione

Questi bit specificano l'istruzione di movimento

Ma occorrono quattro bit di codice istruzioni per identificare uno dei 16 registri:

Codice istruzioni

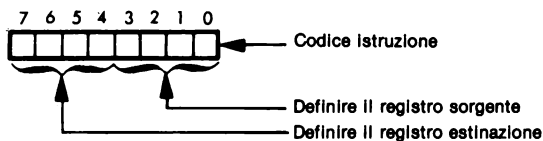
0000	**Registro 0**
0001	**Registro 1**
0010	**Registro 2**
0011	**Registro 3**
0100	**Registro 4**
0101	**Registro 5**
0110	**Registro 6**
0111	**Registro 7**
1000	**Registro 8**
1001	**Registro 9**
1010	**Registro 10**
1011	**Registro 11**
1100	**Registro 12**
1101	**Registro 13**
1110	**Registro 14**
1111	**Registro 15**

Se servono questi quattro bit di codice istruzione per specificare il registro di partenza, e ne servono altri quattro per il registro di destinazione, allora, per un microprocessore ad 8 bit, tutti questi bit di codice istruzione verranno impiegati semplicemente per specificare le istruzioni che spostano i dati fra i registri di partenza ed arrivo:

7 6 5 4 3 2 1 0

Codice istruzione

Definire il registro sorgente

Definire il registro estinazione

Una soluzione adottata da molti progettisti di micro consiste nell'avere un registro "primario", spesso indicato come **accumulatore**; tale accumulatore serve come sorgente o destinazione di ogni spostamento di dati all'interno della CPU.

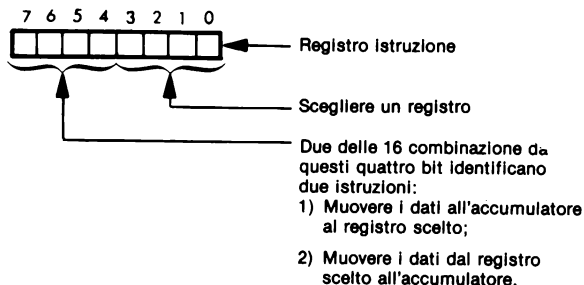
Ora, istruzioni che muovono dati da una locazione ad un'altra entro la CPU devono farlo sempre attraverso l'accumulatore.

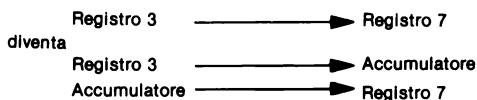
Nel caso di 16 registri, ora ci servono esattamente quattro bit per definire il registro selezionato:

7 6 5 4 3 2 1 0

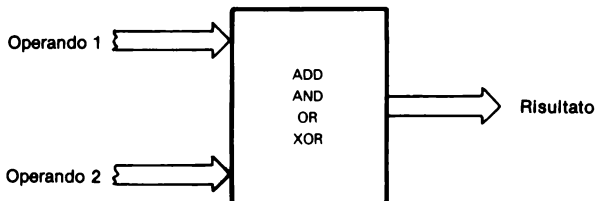
Registro istruzione

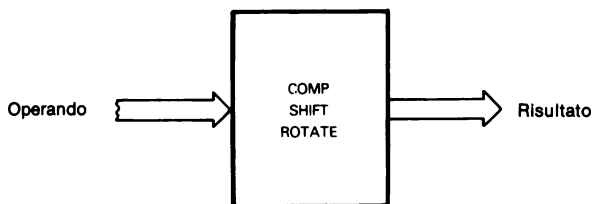
Scegliere un registro

Due delle 16 combinazione da questi quattro bit identificano due istruzioni:
1) Muovere i dati all'accumulatore al registro scelto;
2) Muovere i dati dal registro scelto all'accumulatore.

Da notare che possiamo ancora spostare dati da e fra qualunque registro, ma questa operazione ci porta ad aver bisogno di due istruzioni, anziché una sola, come evidente dalla figura:

Registro 3 → Registro 7
diventa
Registro 3 → Accumulatore
Accumulatore → Registro 7

Esaminiamo ora le istruzioni che specificano le operazioni aritmetiche e logiche. Generalmente esistono due classi di tali operazioni: quelle che richiedono due operandi e quelle che richiedono un solo operando.

Le operazioni di ADD, AND, OR ed OR esclusivo, richiedono due operandi:

Operando 1
Operando 2
ADD
AND
OR
XOR
Risultato

Le operazioni COMPLEMENT, SHIFT e ROTATE richiedono un operando:

Operando
COMP
SHIFT
ROTATE
Risultato

Gli operandi provengono da una delle tre seguenti locazioni:

1) da un registro interno al microprocessore;
2) da una memoria esterna o da un I/O;
3) come costante fornita dalla stessa istruzione.

Consideriamo prima di tutto le istruzioni dei due operandi, per i quali esistono sei possibilità. Usando la notazione [ ], cioè entro parentesi, per contrassegnare "il contenuto", queste possibilità si possono illustrare come segue:

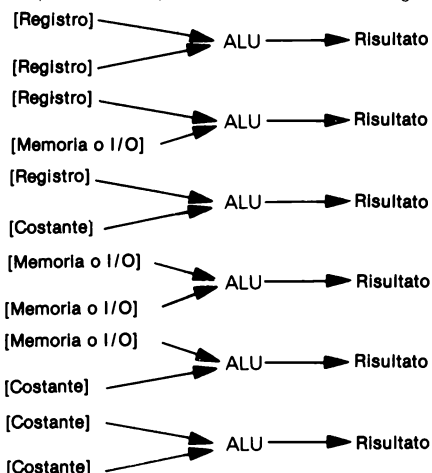
[Registro] → ALU → Risultato
[Registro] ↗

[Registro] → ALU → Risultato
[Memoria o I/O] ↗

[Registro] → ALU → Risultato
[Costante] ↗

[Memoria o I/O] → ALU → Risultato
[Memoria o I/O] ↗

[Memoria o I/O] → ALU → Risultato
[Costante] ↗

[Costante] → ALU → Risultato
[Costante] ↗

Scegliamo arbitrariamente l'operazione ADD ed esaminiamone le possibilità.

Si possono sommare i contenuti di un registro a quelli di un altro. La somma può essere memorizzata in uno dei due, come può essere immagazzinata in un terzo registro, oppure può essere posta in una memoria esterna o locazione di I/O. Mettere il risultato in una "costante" non avrebbe senso, in quanto ciò distruggerebbe la costante.
Si potrebbe sommare il contenuto di un registro ad una memoria esterna o ad un I/O; ancora una volta, la somma potrebbe venir immessa in una delle locazioni dell'operando, o in qualche locazione separata.

Un'istruzione potrebbe anche specificare che il contenuto di due memorie esterne o locazioni I/O vengano sommate, con la somma da memorizzarsi in un registro, in una delle locazioni di operando, o in una diversa memoria esterna o I/O.
In pratica, molto pochi sono i micro che hanno istruzioni che permettono di prelevare due operandi da due locazioni di memoria esterna; ciò in quanto il microprocessore deve eseguire una lettura di memoria esterna per ciascun operando, e ciò risulta in istruzioni generali piuttosto complesse. Per esempio, servirebbero quattro riferimenti di memoria, che si verifichino entro un'esecuzione di istruzioni allo scopo di sommare il contenuto di due locazioni di memoria esterna; il tutto si può indicare:

Clock

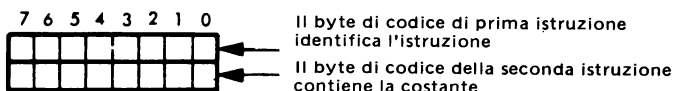
Primo riferimento di memoria	Secondo riferimento di memoria	Terzo riferimento di memoria	Quarto riferimento di memoria
Lettura del codice istruzioni dalla memoria	Lettura del primo operando dalla memoria	Lettura del secondo operando dalla memoria	Scrittura del risultato in memoria

Non c'è nulla di intrinsecamente impossibile nell'avere delle istruzioni complesse, come si è visto sopra; ed in effetti molti minicomputer possiedono istruzioni di questo tipo.

Ma in genere i microprocessori sono più semplici dei microcomputer. Tipicamente i microprocessori si limitano al fetch d'istruzioni, e ad un riferimento di memoria addizionale durante il tempo di esecuzione di un'istruzione.

Perciò un operando può provenire dalla memoria (o da un I/O); oppure il risultato può essere immagazzinato o in memoria o in un I/O, ma non in ambedue.

Un operando può anche essere una costante; tale costante è fornita dal codice di istruzioni. Un'istruzione potrebbe per esempio specificare che un valore costante 3 venga sommato al contenuto di un registro della CPU o di una locazione di memoria esterna.
Un'istruzione che specifichi una costante in pratica include tale costante come parte dell'istruzione, e ciò può essere indicato come segue:

7	6	5	4	3	2	1	0	
								← Il byte di codice di prima istruzione identifica l'istruzione
								← Il byte di codice della seconda istruzione contiene la costante

In realtà una costante non è nient'altro che il contenuto di una locazione nella memoria di programma. Ma, poiché la memoria di programma diventa spesso ROM, la costante diventa realmente una costante, nel senso che essa risiede in una zona di memoria che non può mai più essere modificata.
Le operazioni della ALU che richiedono un singolo operando possono specificare il contenuto di un registro, di una locazione di memoria o di un dispositivo I/O come operandi.

Le istruzioni dei microprocessori non permetteranno di specificare una costante come segnale input per una singola istruzione di operando di ALU, in quanto ciò non avrebbe alcun senso.
Per esempio, un'istruzione per complementare 3 sarebbe inutile: si sa già a cosa corrisponda un'operazione del genere, cosicché questo valore può essere immagazzinato nel primo posto disponibile.

Le istruzioni che modificano il contenuto del contatore di programma cadono in una delle tre seguenti categorie:

1) istruzioni che modificano incondizionatamente il contenuto del contatore di programma;
2) istruzioni che modificano il contenuto del contatore di programma solo quando si incontrano condizioni specifiche identificate da dati appropriati;
3) istruzioni che salvano il contenuto del contatore prima di modificarlo. Queste istruzioni danno l'opportunità di ritornare al punto in cui il contenuto del contatore di programma è stato cambiato; e ciò è possibile reinserendo nel contatore il valore che si era salvato prima di modificarlo.

Mentre le istruzioni che spostano i dati e manipolano la logica della ALU sono autospieganti, le istruzioni che manipolano il contenuto del contatore di programma non sono comprensibili ad un programmatore novizio.
Ciò avviene in quanto queste istruzioni, assieme alle istruzioni di stato, implementano la logica di programmazione, argomento che non ha senso compiuto le prime volte che si studiano i programmi. Per tali motivi, la discussione di questi tipi di istruzione viene rimandata al volume 1°. Lo scopo di questo primo esame sommario dei vari tipi di istruzioni è stato quello di identificare i tipi di operazioni che possono costituire il set di istruzioni di un microprocessore.
Chiaramente, a questo punto, il lettore è ancora lontano dal poter usare tranquillamente il set di istruzioni di un microprocessore; è però pronto ad assimilare i "concetti base" del volume 1°, che sostanzialmente copre gli stessi argomenti dei capitoli IV, V e VI di questo libro, ma con molto maggior dettaglio.
E' alla fine del suddetto volume 1° che il lettore sarà in condizione di partire con l'uso dei microprocessori.

APPENDICE A
CODICE CARATTERE STANDARD

Rappresentazione esadecimale	ASCII (7 bit)	EBCDIC (8 bit)
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
A		
B		
C		
D		
E		
F		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
1A		
1B		
1C		
1D		
1E		
1F		
20	blank	
21	!	
22	"	
23	#	
24	$	
25	%	
26	&	
27	'	
28	(	
29	)	
2A	*	
2B	+	
2C	,	
2D	–	
2E	.	
2F	/	
30	0	

Rappresentazione esadecimale	ASCII (7 bit)	EBCDIC (8 bit)
31	1	
32	2	
33	3	
34	4	
35	5	
36	6	
37	7	
38	8	
39	9	
3A	:	
3B	;	
3C	<	
3D	=	
3E	>	
3F	?	
40	@	blank
41	A	
42	B	
43	C	
44	D	
45	E	
46	F	
47	G	
48	H	
49	I	
4A	J	|
4B	K	.
4C	L	<
4D	M	(
4E	N	+
4F	O	!
50	P	&
51	Q	
52	R	
53	S	
54	T	
55	U	
56	V	
57	W	
58	X	
59	Y	
5A	Z	(
5B	[	$
5C	\	*
5D	]	)
5E		;
5F		∧
60		
61	a	

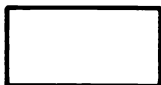
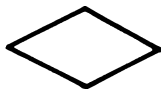
Rappresentazione esadecimale	ASCII (7 bit)	EBCDIC (8 bit)
62	b	
63	c	
64	d	
65	e	
66	f	
67	g	
68	h	
69	i	
6A	j	
6B	k	,
6C	l	%
6D	m	-
6E	n	>
6F	o	?
70	p	
71	q	
72	r	
73	s	
74	t	
75	u	
76	v	
77	w	
78	x	
79	y	
7A	z	
7B		#
7C		@
7D		'
7E		=
7F		"
80		
81		a
82		b
83		c
84		d
85		e
86		f
87		g
88		h
89		i
8A		
8B		
8C		
8D		
8E		
8F		
90		
91		j
92		k
93		l
94		m
95		n
96		o

Rappresentazione esadecimale	ASCII (7 bit)	EBCDIC (8 bit)
97		p
98		q
99		r
9A		
9B		
9C		
9D		
9E		
9F		
A0		
A1		
A2		s
A3		t
A4		u
A5		v
A6		w
A7		x
A8		y
A9		z
AA		
AB		
AC		
AD		
AE		
AF		
B0		
B1		
B2		
B3		
B4		
B5		
B6		
B7		
B8		
B9		
BA		
BB		
BC		
BD		
BE		
BF		
C0		
C1		A
C2		B
C3		C
C4		D
C5		E
C6		F
C7		G
C8		H
C9		I
CA		
CB		

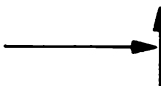
Rappresentazione esadecimale	ASCII (7 bit)	EBCDIC (8 bit)
CC		
CD		
CE		
CF		
D0		
D1		J
D2		K
D3		L
D4		M
D5		N
D6		O
D7		P
D8		Q
D9		R
DA		
DB		
DC		
DD		
DE		
DF		
E0		
E1		
E2		S
E3		T
E4		U
E5		V

Rappresentazione esadecimale	ASCII (7 bit)	EBCDIC (8 bit)
E6		W
E7		X
E8		Y
E9		Z
EA		
EB		
EC		
ED		
EE		
EF		
F0		0
F1		1
F2		2
F3		3
F4		4
F5		5
F6		6
F7		7
F8		8
F9		9
FA		
FB		
FC		
FD		
FE		
FF		

APPENDICE B
SIMBOLI STANDARD DEI DIAGRAMMI DI FLUSSO

Ingresso-Uscita

Elaborazione aritmetica
e movimenti dati

Logica decisionale

Subroutine

Punto di connessione

Frecce di connessione

Punto terminale

NOTE SUGLI AUTORI DEL LIBRO

Adam Osborne è presidente della Osborne and Associates, Inc., una società californiana costituita da consulenti nel settore dei microcomputer.
La Società progetta prodotti basati sui microcomputer o assiste il cliente nella realizzazione dei suoi lavori.
La Osborne and Associates organizza anche seminari sui microcomputer, le loro prospettive e le loro applicazioni immediate. La Società elabora manuali e documentazione tecnica per industrie del settore dei mini e microcomputer.

L. 14.000
(L. 13.208)

20

INTRODUZIONE AI MICROCOMPUTER
VOLUME 0 IL LIBRO DEL PRINCIPIANTE